IL LINGUAGGIO MACCHINA DELLO SPECTRUM

VOL. 2

EDIZIONE ITALIANA

Paul Holmes

GRUPPO EDITORIALE JACKSON

IL LINGUAGGIO MACCHINA DELLO SPECTRUM

VOL. 2

Paul Holmes

GRUPPO
EDITORIALE
JACKSON
Via Rosellini, 12
20124 Milano

COME È FATTO QUESTO LIBRO

Spiegazioni graduali e semplici da seguire.
Moltissimi esempi per fare subito pratica con il linguaggio.
Dizionario dei codici macchina dello Z80.
Esame approfondito delle variabili di sistema.
Stampa di caratteri in doppia altezza.
Come produrre effetti sonori.
La routine della ROM, fra cui quella per l'emissione sonora, la stampa dei caratteri, ecc.
Questo libro conduce il lettore da una conoscenza elementare ad una piena comprensione delle istruzioni per lo Z80, e quindi dello Spectrum.

SOMMARIO

L'AUTORE

Paul Holmes è uno studente diciassettenne che ha iniziato ad operare con i computer due anni fa su di uno Z80. Da allora ha scritto due pacchetti di software per lo ZX81, uno "toolkit" generico ed uno per la grafica, che ora sono commercializzati dalla "JRS Software" in Gran Bretagna, dalla "Softsync" negli USA. Per un certo periodo è stato revisore di software per la rivista "ZX Computing". Recentemente ha preparato vari programmmi che sono stati inclusi nel catalogo Jupiter Cantabs, e si sta ora occupando della redazione di un libro di giochi per lo Spectrum. È originario di Southport, nel Lancashire, ma in seguito ha girato molto la Gran Bretagna; attualmente vive a Sutton Coldfield, nel West Midlands.

RINGRAZIAMENTI E DEDICHE

Un particolare ringraziamento a:
Tim Hartnell, che per primo mi suggerì l'idea di questo libro;
Richard Lawrence, che fece nascere in me l'interesse per il computer;
Liz North, per le uova e toast preparatemi per il pranzo;
Mr. Coulson, il mio insegnante di fisica, per il continuo aiuto durante la preparazione di questo libro.

INTRODUZIONE

Così vi piacerebbe imparare bene il linguaggio macchina? E a che punto siete arrivati? Oh, capisco! Conoscete già le cose elementari ma volete imparare quelle più complicate? Bene, in questo caso avete scelto proprio il libro giusto, in cui si dà per scontata una certa conoscenza del linguaggio macchina. Comunque, anche se ancora non ne capite niente, continuate a leggere: non ho ancora finito!
È proprio vero che il linguaggio macchina è complicato? O non è forse semplice come andare in bicicletta?
In questo libro il mio compito è proprio quello di renderlo semplice come una passeggiata in bicicletta (fino a quando non avremo a disposizione lo schermo piatto non credo di potervi insegnare ad andare in bicicletta).
Un breve appunto, ora, per quelli che sono realmente dei principianti. Come ho già detto, questo libro è fatto per chi conosce già qualcosa; ma anche se questo non è il vostro caso, provate a leggere l'inizio del primo capitolo: se ciò dà qualche ispirazione alla vostra mente senza troppe confusioni, allora comprate pure questo libro. Se invece il vostro cervello vi dà messaggi tipo "frasi prive di senso, allora avete bisogno di iniziare con qualcosa di più semplice, per esempio "IL LINGUAGGIO MACCHINA DELLO SPECTRUM VOL. 1" di James Walsh; questo libro inizia proprio dalle cose più elementari, e procede piano piano fornendo al lettore una conoscenza di base sul linguaggio macchina dello Spectrum.
A questo punto, se avete già acquistato una copia di questo libro, inspirate a fondo e tuffatevi nel primo capitolo. Se invece state solo curiosando in un negozio, decidetevi a comperare questo libro oppure rimettetelo al suo posto: dopo tutto, dove credete di essere, in una biblioteca?

CAPITOLO I

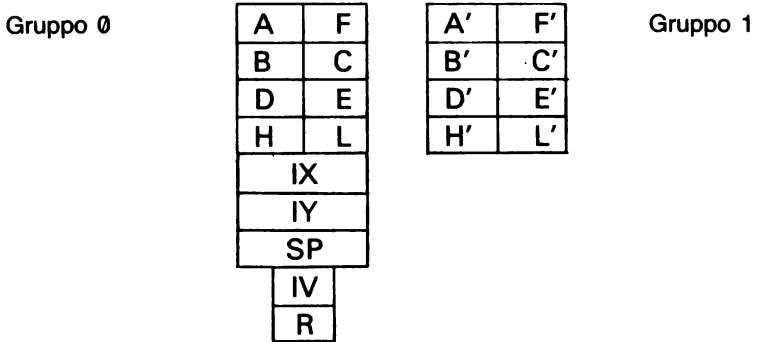
I SALTI

Prima di entrare nel vivo della questione di come si scrive un programma in linguaggio macchina facciamo una breve sintesi di cosa intendiamo fare nei prossimi capitoli e di come usare questo libro.

Come si è accennato nell'introduzione, questo libro non è indirizzato all'utente principiante, ma a qualcuno che già conosce le nozioni più elementari del linguaggio macchina.
Se per caso avete bisogno di un ripasso di tale nozioni, eccovi un breve sommario di ciò che dovrete sapere prima di iniziare.

Il linguaggio macchina è un linguaggio di programmazione comprensibile alla CPU (Central Processing Unit). Esso è rappresentato da una serie di numeri, che vengono tenuti in memoria e che la CPU può capire ed eseguire come istruzioni. Un esempio di ciò è la ROM, un grosso programma nella memoria per sola lettura (Read Only Memory), che sovrintende a tutta l'organizzazione dello Spectrum. Questa ROM è in grado di comprendere le nostre istruzioni in BASIC, di interpretarle e di eseguire le varie routine in linguaggio macchina che effettuano operazioni come PRINT e GOTO.

Il linguaggio macchina è molto più vicino alla macchina vera e propria, molto più rapido e non ha nulla a che vedere con la ROM; il circuito integrato Z80 dello Spectrum è l'unica componente che si occupa del linguaggio macchina. Anche se conservato in memoria, il linguaggio macchina dello Z80 ha una serie di nomi appropriati per le varie istruzioni, cosicché possiamo avere un'idea delle operazioni che vengono eseguite: questi nomi sono chiamati "mnemoniche". Lo Z80 dispone di un certo numero di registri, che costituiscono le fondamenta del linguaggio macchina; si fa uso dei registri proprio come si usano le variabili in BASIC. Ci sono due gruppi di registri, gruppo uno e gruppo zero, ed alcuni registri

speciali che sono l'equivalente delle variabili di sistema. Ecco come si possono rappresentare i registri:

Ciascuna casella contiene un registro. Le caselle più corte possono contenere numeri da 0 a 255, quelle più lunghe possono tenere numeri da 0 a 65535. Unendo due caselle corte adiacenti possiamo ottenere una casella lunga. Per esempio, se uniamo B e C otteniamo il "registro a due byte" chiamato BC. Possiamo usare soltanto i registri A, B, C, D,E, H ed L, i loro equivalenti nel gruppo uno e i registri IX e IY. Gli altri registri sono riservati per scopi particolari al microprocessore Z80

Inoltre, possiamo usare soltanto un gruppo di registri alla volta. Con l'istruzione

"EXX"

si può passare all'altro gruppo e così utilizzare gli altri registri.

Per assegnare un valore numerico ad una variabile in BASIC, scriviamo:

LET A = 20.

In linguaggio macchina dobbiamo invece usare la parola "load", che viene abbreviata in LD, cioè:

LD A, 14.

Il numero 14 non deve essere inteso come i numeri decimali, quelli che usiamo tutti i giorni, bensì come esadecimale, cioè in base 16. Il numero 14 è l'equivalente

Hex (abbreviazione di "Hexadecimal") del numero decimale 20: infatti 1 x 16 + 4 = 20. I Numeri nella colonna di destra vengono conteggiati con il valore di unità, quelli della colonna adiacente a sinistra con valore 16, i successivi (se usiamo un registro a due byte) con valore 256 e quelli della quarta colonna da destra con valore 4096. Poiché nel sistema a base 16 occorrono 16 simboli per rappresentare le cifre, usiamo i dieci numeri da Ø a 9 e poi le prime 6 lettere dell'alfabeto. Ecco la lista dei primi venti numeri:

Hex	Decimal
ØØ	Ø
Ø1	1
Ø2	2
Ø3	3
Ø4	4
Ø5	5
Ø6	6
Ø7	7
Ø8	8
Ø9	9
ØA	1Ø
ØB	11
ØC	12
ØD	13
ØE	14
ØF	15
1Ø	16
11	17
12	18
13	19
14	2Ø

In un byte di memoria possono essere tenute due cifre esadecimali, e quindi possiamo usare i numeri decimali da Ø a 255.

Con i registri possiamo eseguire moltissime operazioni. Ci sono istruzioni come LD per caricare un valore in un registro, altre come ADD per sommare due registri. Come ho già accennato, un programma viene conservato in memoria

come una serie di codici numerici. Per esempio, l'istruzione "carica in A un dato" consta di due numeri esadecimali:

LD A, d3E d

Il primo numero è l'esadecimale 3E, che indica che stiamo caricando un dato in A, e "d" (che può essere un numero qualsiasi da un byte) è il dato che deve essere messo in A. Nel caso di un caricamento nel registro a due byte, per esempio in BC, abbiamo bisogno di specificare due byte di dati, ma stranamente essi devono essere indicati "alla rovescia". Ecco come si può caricare il numero esadecimale 156A in BC:

LD BC, 156A01 6A 15

Il primo bite è 01, che indica che stiamo compiendo un'istruzione di caricamento nel registro BC; poi c'è il "byte meno significativo" di dati, cioè le due cifre di destra del numero da caricare; infine, il numero 15 Hex è il "byte più significativo" di 156A. Specifichiamo sempre per primo il byte meno significativo: questa è una convenzione, e dovremo sempre seguirla.
Esistono molte altre istruzioni, che sono tutte riportate nell'Appendice D. Le istruzioni che dovreste già conoscere sono le istruzioni di caricamento, quelle aritmetiche come ADD e SUB, RET e inoltre dovreste sapere qualcosa dello stack (sebbene ciò venga spiegato in questo capitolo).

Se incontrate in un programma un'istruzione che non capite e che non viene spiegata nel testo, potrete trovarne una breve definizione nell'Appendice A.

In tutto questo libro ho fatto uso di un programma di caricamento esadecimale ("hex-loader") scritto in BASIC, con cui immettere nel computer i programmi in linguaggio macchina. Se avete a disposizione un programma di "editor" per il linguaggio macchina, certamente vi converrà usarlo. Tutti i listati del libro includono sia gli indirizzi dei codici che i codici operativi stessi. Si noti anche che nei listati i numeri sono sempre esadecimali, mentre nel testo ci sarà sempre dopo ogni numero la parola "esadecimale" (o "Hex") oppure decimale per chiarire di che numero si tratti. Se possedete un assembler potrete inserire i programmi usando le mnemoniche che verranno indicate, ma assicuratevi che il vostro assembler accetti le mnemoniche standard dello Z80 Zilog e che possa assemblarle in linguaggio macchina agli indirizzi specificati.

Nel secondo capitolo inizieremo a vedere i vari metodi per i salti in un programma. Esamineremo poi in che modo un computer possa prendere decisioni; inoltre a questo capitolo ci sarà un programma che consentirà di usare nello Spectrum

caratteri ad altezza doppia del normale. Il terzo capitolo sarà interamente dedicato ai "bit" di ogni byte, esaminando come sia possibile porre questi bit a uno o a zero, come eseguire vari test, e si vedrà come esempio il file degli attributi.

Il terzo capitolo si chiuderà con un programma che consente allo Spectrum di diventare una macchina da scrivere intelligente. Il quarto capitolo si occupa delle istruzioni logiche XOR, AND e OR, usando anche qui il file degli attributi per dare qualche esempio. Il capitolo successivo sarà dedicato alle istruzioni di "rotazione" e al loro utilizzo; inoltre, nello stesso quinto capitolo si tratterà brevemente del sistema "Binary Coded Decimal" (decimali a codifica binaria) e di come usarlo con le istruzioni dello Z80.

Il sesto capitolo riguarderà in dettaglio le cosiddette "porte" e si discuterà il modo in cui il computer possa comunicare con il mondo esterno ed il modo in cui ci si possa servire di queste comunicazioni. Si spenderà molto tempo per vedere come si possano generare effetti sonori con lo Spectrum. Il capitolo finale intitolato "Posso interrompere?", si occuperà di un argomento che collega fra loro discorsi di software e di hardware: il sistema degli "interrupt". Esso verrà spiegato con l'ausilio di materiale della ROM e si descriveranno i due tipi di "interrupt" e i tre modi di "interrupt".

Alla fine di questo libro ci sono quattro appendici che riportano: le mnemoniche esadecimali, decimali ASC11 e dello Z80; il modo in cui le istruzioni dello Z80 influenzano i segnali ("flag"); le variabili di sistema, con altre spiegazioni aggiuntive; e la spiegazione del funzionamento di ciascuna delle istruzioni dello Z80.

Ora che abbiamo detto tutto ciò, possiamo iniziare con il primo argomento, i salti in linguaggio macchina, una parte importantissima in ogni programma in linguaggio macchina.

Quando si scrive un programma in BASIC, succede spesso di volersi muovere da un punto ad un altro del programma in linguaggio macchina.
Quando si scrive un programma in BASIC, succede spesso di volersi muovere da un punto ad un altro del programma. Possiamo farlo semplicemente con la frase GOTO. È abbastanza utile il fatto che non deve necessariamente essere seguita da un numero di riga: ci può essere anche un'espressione algebrica, per es.:

GOTO 10 * A + 100

Questo è un sistema molto versatile per effettuare i salti.
In effetti la parola "salto" viene associata più spesso ai movimenti nell'ambito di un programma in linguaggio macchina, piuttosto che a quelli in un programma in BASIC. Essa è comunque una parola molto adatta per qualsiasi linguaggio in cui

è consentito muoversi da una parte all'altra del programma. In linguaggio macchina le cose sono un po' diverse. Le espressioni algebriche non esistono, ma in compenso ci sono due diversi tipi di istruzioni per dire al microprocessore Z80 dello Spectrum di continuare l'esecuzione del programma da un altro punto.

Come al solito sembra che il linguaggio macchina abbia delle soluzioni più complesse per ogni cosa. Dovremo quindi vedere prima il nostro vecchio amico, il "Program Counter" (contatore di programma), per rendere più chiare le spiegazioni successive.

All'interno di un microprocessore Z80 ci sono moltissimi registri, più che in numerose altre CPU ("Central Processing Unit", unità centrale di elaborazione), i quali ci aiutano nei nostri vari problemi di programmazione. Alcuni di essi appartengono esclusivamente alla CPU per utilizzo proprio, un po' come le variabili di sistema del BASIC. Uno di questi registri è il Program Counter.

Il Program Counter dice alla CPU a che punto ci si trova nell'esecuzione di un programma. Considerate il seguente programma:

```
8000 3E10      LD   A,10
8002 0608      LD   B,08
8004 80        ADD  A,B
8005 C9        RET
```

Alla sinistra sono indicati gli indirizzi, inziando da 8000: questo è il punto da cui inizia il programma memorizzato. Scorrendo le effettive istruzioni del programma, vediamo che in A viene caricato il numero 10 esadecimale. Assicuratevi di avere ben fisso in mente che stiamo lavorando in esadecimale, altrimenti vi confonderete moltissimo! Poi 08 Hex viene caricato nel registro B. A e B vengono sommati. Infine l'istruzione RET ci riporta al BASIC.

Vediamo ora il Program Counter. Man mano che la CPU "raccoglie" le varie istruzioni, il Program Counter mantiene il controllo dell'indirizzo al quale ci troviamo in ogni momento. All'inizio, quando diamo il comando per l'esecuzione del programma, il Program Counter, detto in breve PC, contiene il valore esadecimale 8000, l'indirizzo cioè in cui si trova la prima istruzione. La CPU capisce che deve caricare nell'accumulatore un "valore diretto", e quindi muove in avanti PC all'indirizzo successivo 8001, dove trova il valore 10 Hex. Questo valore viene caricato nell'accumulatore e poi PC viene di nuovo spostato in avanti all'indirizzo 8002 per prendere l'istruzione successiva. Si continua così finché non viene dato il comando di muoversi ad un altro punto. Nel nostro esempio si continuerà così fino all'istruzione RET, ma la esamineremo più avanti in modo più dettagliato.

Vi chiederete senz'altro "come si fa per mandare PC in un altro punto?", quindi passo subito a spiegarvi questo argomento. Dobbiamo usare l'istruzione JP seguita da un indirizzo, ossia il posto dove andare. Questa è veramente simile all'istruzione GOTO, tranne che non si possono usare espressioni; si possono usare alcuni registri, ma lo vedremo più avanti.
Guardate ora questo programma abbastanza inutile:

```
8000 3E10     LD   A,10
8002 0601     LD   B,01
8004 80       ADD  A,B
8005 C30480   JP   8004
```

Ciò che viene fatto è fissare nell'accumulatore il valore 10 Hex e nel registro B il valore 1 Hex, poi sommare B ad A. A questo punto l'istruzione JP fa sì che si vada all'indirizzo 8004, dove di nuovo vengono sommati A e B. Si procede così all'infinito, oppure finché non si toglie la corrente!

In pratica succede che quando incontra l'istruzione JP, la CPU prende i due byte successivi e li carica in PC. Ora PC contiene un valore diverso dal successivo indirizzo e la CPU continuerà l'esecuzione da un'altro punto della memoria, in questo caso 8004. Si noti come viene scritto 8004 nel programma: 04 prima di 80. Attenetevi sempre a questo sistema "alla rovescia" di scrivere i numeri quando sono formati da due byte. Nel caso dell'istruzione JP, essa deve sempre essere seguita da due byte.

Per esempio, se volete saltare all'indirizzo 0001, non potete tralasciare i primi due zeri:

```
8000 C30100    JP    0001
```

Considerate il seguente programma:

```
8000 C30680    JP 8006
8003 C30980    JP 8009
8006 C30380    JP 8003
8009 C30380    JP 8003
800C C9        RET
800D C30C80    JP 800C
```

Si raggiungerà mai l'istruzione RET per tornare al BASIC?

Avrete ormai capito che il codice per l'istruzione JP è C3, seguito da due byte per dire alla CPU a quale indirizzo saltare. Esistono altre forme per l'istruzione JP, la maggior parte delle quali saranno esaminate nel secondo capitolo; alcune però le vedremo subito, cioè:

JP (HL)
JP (IX)
JP (IY)

In BASIC è possibile dire:

GOTO A

Anche in linguaggio macchina, seppure con certe limitazioni, possiamo operare in modo analogo, usando il registro HL oppure i registri indice. Il registro IY non può essere usato, altrimenti si potrebbe creare confusione nei lavori dello Spectrum, anche se non in modo permanente, e si potrebbero perdere i programmi.
JP (HL)

```
6FF9 010200    LD    BC,0002
6FFC 210070    LD    HL,7000
6FFF E9        JP    (HL)
7000 09        ADD   HL,BC
7001 E9        JP    (HL)
7002 C9        RET
```

Fa sì che si salti all'istruzione contenuta nell'indirizzo in HL. Per esempio, se HL contenesse 700C (Hex) si salterebbe all'indirizzo 700C.

Anche questo è un programma inutile, ma provate a seguirne il funzionamento:

6FF9 : si carica 2 in BC.
6FFC : si carica 7000 in HL.
6FFF : si salta al valore contenuto in HL, che in questo momento è 7000.

7000 : BC viene sommato ad HL, che così contiene 7000+2 cioè 7002.
7001 : un'altro salto viene fatto all'indirizzo in HL, questa volta 7002 (inutile perchè è l'indirizzo successivo).
7002: RET, e si torna al BASIC.

Proveremo adesso a scrivere nello Specturm alcuni brevi programmi. Il primo sarà quello che abbiamo appena visto, così vi convincerete che funziona.

Per inserire un programma in linguaggio macchina nello Spectrum, abbiamo bisogno di un piccolo programma in BASIC, che tramite l'istruzione POKE lo

scriva al di sopra del RAMTOP. Scrivete le seguenti istruzioni:

```
  10 LET indirizzo=28665
  15 READ a$
  20 LET byte=0
  30 FOR i=1 TO 0 STEP -1
  40 LET a=CODE a$-55: IF a$<":"
 THEN LET a=a+7
  50 LET byte=byte+a*16↑i: LET a
$=a$(2 TO ): NEXT i
  60 POKE indirizzo,byte: LET in
dirizzo=indirizzo+1: IF LEN a$=0
 THEN GO TO 15
  70 GO TO 20
  80 DATA "010200","210070","E9"
,"09","E9","C9"
```

Questo programma converte i codici esadecimali in numeri decimali e poi li mette con POKE in un punto della RAM a nostra scelta. La linea 10 contiene

LET indirizzo = 28665

perché la prima istruzione del programma in linguaggio macchina che vogliamo scrivere deve trovarsi in 6FF9 (cioè 28665 decimale). I codici operativi, in esadecimale, sono scritti in una frase DATA alla line 80. Abbiamo abbassato il RAMTOP, usando il comando CLEAR, fino all'indirizzo 28600; così esso si trova più in basso del necessario, ma è bene lasciare sempre spazio per vari movimenti: per esempio si può voler cambiare o modificare un programma. Adesso scrivete RUN e il programma si arresterà con il messaggio:

Out of DATA Line 80

Nessun problema, è solo che non abbiamo messo un controllo per la fine dei dati: non c'è stato alcun danno.
Il programma in linguaggio macchina viene caricato in dieci byte della RAM a partire da 6FF9, ovvero 28665 decimale. Adesso possiamo farlo eseguire, scrivendo:

PRINT USR 28665

Sul video apparirà il numero 2.

In questo modo sapremo due cose. La prima è che il programma è riuscito a raggiungere l'istruzione RET contenuta nell'indirizzo 7002 Hex. Possiamo esserne certi, perché altrimenti il computer sarebbe rimasto bloccato in un ciclo infinito e avremmo dovuto tirar via la spina! La seconda cosa che notiamo è che viene

stampato un 2 sul video. Ciò che succede è che quando termina l'esecuzione di un nostro programma in linguaggio macchina, giungendo all'istruzione RET, lo Spectrum riporta sul video l'ultimo valore tenuto in BC.

Se guardiamo il programma vediamo che c'è l'istruzione Ld BC, 0002 che come sapete fa sì che il registro BC contenga il valore 2. Ecco perché, quando usiamo PRINT con l'istruzione USR, comparirà un numero sul video.
Possiamo evitare che ciò avvenga mediante l'istruzione LET e usando una variabile "dummy": questa è una variabile che non serve ad alcuno scopo utile, tranne quello di associarsi con l'istruzione LET.
Provate:

LET dummy = USR 28665

Questa volta non succede nulla sul video, però il programma ha funzionato lo stesso.
Per verificarlo scrivete:

PRINT dummy

e, sorpresa! esce un 2 sul video.

A volte si può voler spostare un programma ad un altro indirizzo nella memoria. Per esempio, supponiamo di avere un programma scritto per stare proprio in cima alla RAM su uno Spectrum da 16K byte.
Poi un amico desidera servirsene, ma questa volta caricandolo in cima alla RAM di uno Spectrum da 48K byte.
Usando il nostro programma di caricamento in BASIC, la cosa ovvia da fare è cambiare il valore assegnato alla variabile "adress" alla linea 10. Ma se abbiamo usato l'istruzione JP nel programma, ci sarà anche qualcos'altro da cambiare.

```
7FF0 0608      LD    B,08
7FF2 3E4C      LD    A,4C
7FF4 80        ADD   A,B
7FF5 C3FF7F    JP    7FFF
```

Il programma qui sopra è quello scritto per lo Spectrum da 16K. Esso arriva proprio fino all'indirizzo 7FFF dove si incontra l'istruzione RET e si torna al BASIC.

Vediamo ora cosa succede se lo spostiamo semplicemente ad un indirizzo più in alto della memoria dello Spectrum:

```
FFF0 0608      LD   B,08
FFF2 3E4C      LD   A,4C
FFF4 80        ADD  A,B
FFF5 C3FF7F    JP   7FFF
```

La somma di 08 più 4C continua ad essere eseguita normalmente però poi si salta ad 7FFF.
Questo byte può contenere un qualsiasi vecchio residuo di altri valori dello Spectrum del vostro amico. Perciò dovremo modificare l'indirizzo per il salto, portandolo ad FFFF: solo così il programma potrà funzionare perfettamente.
Ciò è abbastanza semplice, ma supponete di avere invece un programma più lungo, con una cinquantina di istruzioni di salto. Per modificarle tutte ci vorrebbe del tempo.
Abbiamo bisogno invece di un programma che sia facilmente trasferibile da un indirizzo ad un altro, cioè che possa essere messo quasi ovunque senza modifiche.

In linguaggio macchina esiste una speciale istruzione che rende possibile questa trasferibilità. Ci sono alcune limitazioni aggiuntive rispetto all'istruzione JP, ma resta comunque un'istruzione molto versatile. Per usarla dobbiamo prima capire la convenzione del "complemento a due".

Il salto stesso viene chiamato "salto relativo" ("Jump Relative" abbreviato in JR), e viene seguito da un numero con cui si specifica quanti byte devono essere saltati e se si deve saltare in avanti o all'indietro. Se si deve saltare all'indietro, si usa un numero negativo: è proprio a questo proposito che dobbiamo imparare la convenzione del "complemento a due".

Naturalmente la CPU tratta solo numeri positivi, ma alcune volte, come nel caso dell'istruzione JR, può riconoscere come negativi certi numeri da un solo byte. Per formare un numero negativo, per esempio meno 10 (Hex), lo si sottrae da 100 (Hex):

100 Hex - 10 Hex = F0 Hex (ovvero - 10 Hex)
256 dec - 16 dec = 240 dec (ovvero -16 dec)

Il calcolo della seconda riga è esattamente identico a quello della prima, ma in decimale. Probabilmente vi risulterà più facile fare le sottrazioni in decimale, perciò

dovrete fare qualche conversione. A questo scopo potrete usare l'Appendice A del manuale Sinclair, oppure l'Appendice A di questo libro.

In un insieme di istruzioni per lo Z80 c'è un'istruzione che fa automaticamente questo calcolo. L'istruzione è NEG, e serve a rendere negativo un numero positivo contenuto nell'accumulatore, proprio come abbiamo fatto prima manualmente. Per vedere il funzionamento di questa istruzione, consideriamo un breve programma.

Scrivete di nuovo il programma in BASIC listato sopra (a meno che non lo abbiate ancora in memoria), scrivendo però:

CLEAR 28600.

Ora eccovi il programma in linguaggio macchina che useremo:

```
7000 3E10        LD   A,10
7002 ED44        NEG
7004 4F          LD   C,A
7005 0600        LD   B,00
7007 C9          RET
```

Si noti che l'istruzione NEG richiede due byte: ED (Hex) è il prefisso, 44 (Hex) è il codice operativo. Per caricare questo programma nello Spectrum sostituite la linea 80, l'istruzione DATA, con la seguente:

```
  80 DATA "3E10","ED44","4F","06
00","C9"
```

Cambiate poi la linea 10 con:

```
10 LET indirizzo=28672
```

Adesso scrivete RUN, e infine PRINT USR 28672. Come nelle nostre previsioni, sul video comparirà 240, cioè la risposta in numero decimali, la stessa che abbiamo trovato prima. Vediamo come funziona questo programma:

7000: 10 Hex viene caricato in A.
7002: viene cambiato di segno, cioè meno 10.
7004: carichiamo A in C, così possiamo vedere il risultato sul video.
7005: il byte B più significativo viene azzerato perché non è necessario per numeri minori di 256.
7007: RET ci riporta al BASIC.

Considerate ora questo programma:

```
7000 0E00      LD   C,00
7002 0601      LD   B,01
7004 1804      JR   700A
7006 0E10      LD   C,10
7008 0600      LD   B,00
700A C9        RET
```

Prima viene azzerato C e in B viene caricato il numero 01 Hex: così BC contiene 0100. Poi viene effettuato un salto relativo: 04 significa che si devono saltare i 4 byte successivi, e perciò le istruzioni LD C, 10 e LD B, 00 non vengono prese in considerazione, passando direttamente all'istruzione RET. Il numero che segue l'istruzione JR, in questo caso 04, opera sempre da quel byte quando è positivo.

Se ci fosse stato il numero 08, si sarebbero saltati gli otto byte successivi.

Guardate ora questo programma:

```
7000 C9        RET
7001 3E10      LD   A,10
7003 0606      LD   B,06
7005 80        ADD  A,B
7006 18F8      JR   7000
7008 00        NOP
```

Se iniziamo ad eseguirlo dall'indirizzo 7001, in A viene caricato 10 (Hex), in B viene caricato 06, i due numeri vengono sommati e si giunge al salto relativo. Questa volta il numero successivo è negativo: si devono saltare otto byte all'indietro. Partendo dall'istruzione NOP e muovendosi all'indietro di otto byte, si giunge all'istruzione RET, che ordina alla CPU di tornare al BASIC.
Controlliamo:

265 — 8 = 242; convertiamo i numeri in esadecimale: 242 = F8 Hex.

F8 segue l'istruzione JR, quindi va tutto bene.
Usando il complemento a due, possiamo rappresentare con un byte i numeri positivi da 0 a 126 decimale ed i numeri negativi da -1 a -127 decimale. (Sì, lo zero conta come positivo.) In un lungo programma non saremo quindi in grado di usare sempre e soltanto l'istruzione JR, dato che non potremo fare salti superiori a 127 byte in avanti o all'indietro. Così non potremo rendere completamente trasferibili tutti i programmi.

Considerate:

```
7000 18FE      JR   7000
7002 C9        RET
```

Dovrebbe esservi chiaro che FE (Hex) viene itnerpretato come -2. Questo programma quindi non fa altro che tornare all'infinito all'istruzione JR, oppure finché non si stacca la corrente.
Se non vi dispiace dovervi scrivere di nuovo il programma di caricamento in BASIC, provate a far eseguire quest'ultimo programma.
Cambiate la linea 80 con:

```
80 DATA "18FE","C9"
```

Scrivete RUN e poi PRINT USR 28672.

Sembra che non succeda nulla!

In effetti però il computer è bloccato in un ciclo infinito! Provate a schiacciare il tasto BREAK. Di nuovo non succede nulla. Ricordatevi che il tasto BREAK non funziona in linguaggio macchina. Si può fare per conto proprio nel programma il controllo della pressione di questo tasto, e impareremo come farlo più avanti in questo libro. Per il momento potete soltanto staccare la spina e ricominciare.

Qualche volta in un programma può essere necessario usare più volte una stessa routine. Per evitare di doverla riscrivere ogni volta che ce n'è bisogno si può usare una subroutine. Le subroutine in linguaggio macchina sono molto simili a quelle che si usano in BASIC, con GOSUB e RETURN. Anche in linguaggio macchina c'è RETURN scritto RET, ma al posto di GOSUB si deve usare CALL.

Quando lavoriamo con il linguaggio macchina dello Spectrum, il nostro programma è in effetti una subroutine chiamata in eescuzione dalla ROM; di conseguenza, quando vogliamo che termini l'esecuzione e che si ritorni alla ROM (cioè al BASIC), dobbiamo usare l'istruzione RET.

L'istruzione CALL richiede un indirizzo diretto, per esempio 7000, che viene caricato direttamente in PC, non calcolato in modo relativo come in JR. Non esiste una forma di chiamata relativa, il che impedisce anche di avere programmi completamente rilocabili.

Il seguente programma somma due numeri a due byte contenuti in BC e DE e restituisce il risultato in HL:

```
7000 60         LD    H,B
7001 69         LD    L,C
7002 19         ADD   HL,DE
7003 C9         RET
```

Esso è scritto come subroutine, quindi ecco un programma che può utiizzarlo:

```
7004 014C2A     LD    BC,2A4C
7007 117E4D     LD    DE,4D7E
700A CD0070     CALL  7000
700D 44         LD    B,H
700E 4D         LD    C,L
700F C9         RET
```

Il numero 2A4C Hex viene caricato in BC e 4D7E Hex in DE; poi si chiama con CALL la subroutine di somma; infine in risultato viene caricato in BC in modo da poter ottenere il risultato stampato sul video in decimale quando torniamo al BASIC. Facciamolo ora eseguire modificando la linea 10:

```
10 LET indirizzo=28672
```

Cambiate anche la linea 80:

```
  80 DATA "60","69","19","C9","0
14C2A","117E4D","CD0070","44","4
D"
```

Adesso scrivete RUN e poi PRINT USR 28672. Sul video comparirà 30666, cioè il risultato in decimale.

Ci sono alcune subroutine che sono utilmente applicabili in molti programma: per esempio quella del capitolo sette che fa il controllo del tasto BREAK. La ROM stessa contiene numerose subroutine molto utili , ne vedremo due in seguito.

Come forse avrete già notato, il codice esadecimale per un'istruzione CALL non condizionata è CD, seguito dall'indirizzo diretto, cioè due byte; per esempio:

CALL 70C0 - CD C0 70

Ci sono altri tipi di istruzioni CALL detti chiamate condizionate, ma li esamineremo nel secondo capitolo.
Un'altro modo per effettuare una chiamata è mediante l'istruzione RST.
Essa differisce dalle altre chiamate in quanto permette di accedere soltanto alle subroutine poste ai seguenti indirizzi:

```
0000
0008
0010
0018
0020
0028
0030
0038
```

Questi indirizzi sono tutti contenuti nella ROM. In questo modo la chiamata è più rapida che con l'istruzione CALL, e si occupa meno spazio in quanto è usato un codice diverso per ogni indirizzo di RST. RST è l'abbreviazione di RESTART.

```
RST 00  —  C7
RST 08  —  CF
RST 10  —  D7
RST 18  —  DF
RST 20  —  E7
RST 28  —  EF
RST 30  —  F7
RST 38  —  FF
```

Vi chiederete sicuramente come mai vi stia dicendo tutto ciò, poiché se si tratta di indirizzi nella ROM non li potremo usare per le nostre subroutine.

Tuttavia nella ROM c'è una sobroutine utilissima, che può essere chiamata con RST 10. Si tratta della routine di stampa sul video, con cui si possono stampare tutti i caratteri lampeggianti, brillanti e colorati.

Questa subroutine è una vera miniera per noi programmatori in linguaggio macchina e ci risparmia moltissimi fastidi legati alla stampa. Prima si deve caricare nel registro A il carattere che si vuole stampare e poi, usando l'istruzione RST 10, lo si può far stampare sul video. Per esempio, supponiamo di voler stampare "CIAO":

```
LD   A,43
RST  ,10
LD   A,49
RST  ,10
LD   A,41
RST  ,10
LD   A,4F
RST  ,10
RET
```

Anzitutto nell'accumulatore viene caricato il numero 43 esadecimale: guardando nella tabella dell'Appendice A del manuale Sinclair o di questo libro potrete verificare che questo è il codice per il carattere "C". Poi RST 10 fa sì che la CPU esegua una subroutine che inzia all'indirizzo 10 Hex, per dirla in termini tecnici. Successivamente si ripetono le stesse operazioni per gli altri caratteri della parola "CIAO". Prima ho accennato al colore; vediamo ora come "colorare" le parole e qualche altra cosa. Guardate di nuovo l'Appendice A e vedrete che il codice di controllo per "l'inchiostro" (INK) è 10 Hex. Se inviamo questo codice alla routine di stampa, seguito da un numero fra 00 e 07, si potrà selezionare un colore d'"inchiostro" appropriato! Proviamo ora a stampare il nome Peter in blu:

```
7000 3E10      LD    A,10
7002 D7        RST   10
7003 3E01      LD    A,01
7005 D7        RST   10
7006 3E50      LD    A,50
7008 D7        RST   10
7009 3E65      LD    A,65
700B D7        RST   10
700C 3E74      LD    A,74
700E D7        RST   10
700F 3E65      LD    A,65
7011 D7        RST   10
7012 3E72      LD    A,72
7014 D7        RST   10
7015 C9        RET
```

Si inizia inviando il codice 10 Hex tenuto nell'accumulatore alla routine di stampa, cosicché il successivo codice da inviare deve essere il codice di un colore, nel nostro caso è 01 corrispondente al blu. Se non si manda un codice valido (cioè compreso fra 00 e 09 inclusi), comparirà il seguente messaggio di errore:

K Invalid colour.

Nel seguito del programma vi accorgerete, facendo riferimento all'Appendice A, che ci sono i codici relativi ai caratteri della parola "Peter". In particolare si noti che il codice per l'istruzione RST 10 è D7. Forse vi sembrerà che il programma sia molto lungo e complicato per la sola stampa della parola "Peter" in colore blu, in effetti questo è il metodo più lento per farlo (lento nel senso del tempo impiegato per scrivere il programma, ma è molto rapido come esecuzione). Prima o poi troverete un metodo più semplice, altrimenti potrete usare la mia routine che troverete descritta più avanti. Facciamo ora eseguire il "Peter" in blu: cambiate la linea 10 del programma di caricamento in:

```
10 LET indirizzo=28672
```

a meno che non sia già così, e la linea 8Ø con:

```
80 DATA "3E10","D7","3E01","D7
","3E50","D7","3E65","D7","3E74"
,"D7","3E65","D7","3E72","D7","C
9"
```

Poi, dopo aver controllato che sia tutto esatto, scrivete RUN e infine:

PRINT AT Ø,Ø;: RANDOMIZE USR 28672.

La parola "Peter" dovrebbe apparire in blu sul video. Considerate il terzo blocco di dati nella linea 8Ø del loader, "3E01": provate a cambiare il codice Ø1 con quello di un altro colore, per controllare che funzioni bene. Se lo fate ricordatevi di far girare di nuovo il loader prima di scrivere la linea con il comando USR. Io non affermo certo di conoscere tutto dello Spectrum: una delle cose per cui non ho trovato ancora un valido motivo è che quando si tralascia l'istruzione PRINT AT ed il comando USR viene eseguito da solo, la parola "Peter" in blu compare e scompare. Provate anche voi:

RANDOMIZE USR 28672

Visto cosa succede? Ora provate a vedere cosa succede con i seguenti comandi:

i CLS : RANDOMIZE USR 28672
ii PRINT TAB 1Ø : RANDOMIZE USR 28672
iii PRINT' : RANDOMIZE USR 28672
iv PRINT; : RANDOMIZE USR 28672
v PRINT, : RANDOMIZE USR 28672

Ci sono molte altre cose che si possono fare con la routine di stampa: si può far lampeggiare la parola oppure renderla più brillante. Anche per questo fate riferimento a quell'inesauribile fonte di informazioni che è l'Appendice A e cercate il codice per "FLASH", lampeggiamento. Con un po' di fortuna troverete che è 12 Hex. Se non ci siete arrivati c'è un errore di stampa oppure avete bisogno di un paio di occhiali! Ora, per inserire il lampeggiamento, facciamo seguire a 12 il comando Ø1; per disinserirlo invece dopo il 12 usiamo il codice ØØ. Analogamente con il comando per "BRIGHT", alta luminosità: il suo codice è 13 Hex e useremo Ø1 e ØØ rispettivamente per inserire e disinserire questo attributo della stampa. Proviamo ora a rendere lampeggiante e più brillante la parola "Peter" in blu.

Inserite la seguente sezione di dati esadecimali all'inizio della linea 80 (subito prima di "3E10"):

"3E12", "D7", "3E01", "D7", "3E13", "D7", "3E01", "D7"

Dovrebbe risultarvi chiaro che il 12 è per il lampeggiamento e il 13 per l'alta luminosità. Ricordatevi di verificarlo, poi fate girare il loader e infine scrivete:

PRINT AT 0,0;: RANDOMIZE USR 28672

Esistono molti altri codici di cotnrollo che possiamo usare: ecco la lista completa.

Codice	Descrizione
06	Questo codice muove la posizione di stampa alla colonna 0 o 16 a seconda di qual'è la seguente, proprio come con una virgola nell'istruzione PRINT. Il codice 06 va usato da solo.
08 sinistra 09 destra 0A giù 0B su	Con questi quattro codici vi aspettereste di poter muovere il cursore in basso, in alto, etc.; in effetti però solo 08 (sinistra) funziona, mentre gli altri fanno comparire un "?". Il codice (sinistra) può essere usato per la sopra scrittura, descritta nel manuale Sinclair.
10 INK	abbiamo già usato questo codice, ma giusto per riepilogare ricordiamo che esso deve essere seguito da un valido codice di colore. (INK) da 00 a 09 per cambiare il colore di stampa
11 PAPER	questo codice, proprio come il 10 (INK) fissa il colore per tutta la pagina. Anche questo deve essere eseguito da un codice di colore valido
12 FLASH	il codice 12 inserisce il lampeggiamento se seguito da 01, lo disinserisce se seguito da 00. Il codice 08 può essere usato per la stampa "trasparente", cioè stampa che è identica a ciò che stava sotto (una spiegazione completa può essere trovata nel capitolo 16 del manuale Sinclair)
13 BRIGHT	cambia la luminosità nello stesso modo descritto per il lampeggiamento (codice 12)
14 INVERSE	inserisce o disinserisce la stampa inversa proprio come per il comando BRIGHT
15 OVER	questo codice inserisce o disinserisce il modo di sovrascrittura. Usato in combinazione con il codice 08 (cursore a sinistra), può servire per scrivere dei caratteri sopra ad altri. Deve essere segiuto da 01 o 00 a seconda della necessità.

CAPITOLO 2

I SALTI CONDIZIONATI

Sappiamo ora come fare i salti, i salti relativi e come chiamare una subroutine e fare ritorno al programma chiamante. Il successivo argomento da esaminare è quello delle decisioni: dopo tutto un computer viene definito come una macchina in grado di prendere decisioni logiche.

DECISIONI, DECISIONI

La CPU prende le sue decisioni sulla base di un solo registro, il registro F ovvero quello dei segnali (Flag). Vediamo com'è strutturato questo registro nei suoi singoli bit:

BIT	0	C	segnale di riporto ("carry flag")
	1	N	segnale di addizione/sottrazione
	2	P/V	segnale di parità/overflow
	3	—	NON UTILIZZATO-SEMPRE PARI A ZERO
	4	H	segnale di riporto intermedio ("half carry flag")
	5	—	NON UTILIZZATO-SEMPRE PARI A ZERO
	6	Z	segnale di azzeramento ("zero flag")
	7	S	segnale del segno ("sign flag").

Come vedete ci sono solo sei segnali, dato che due bit non vengono utilizzati. Ciascun bit viene indicato con una lettera, come abbreviazione del suo significato. Possiamo tralasciare il segnale di addizione/sottrazione (N) ed il segnale H, dato che la CPU non può prendere decisioni sulla base di questi bit, e non ne avrebbe nemmeno necessità.

I segnali vengono modificati dopo certe istruzioni in modo prestabilito. Di solito cambiano quando viene eseguita un'operazione con l'accumulatore, come una sottrazione o un'addizione; cambiano anche quando si incrementa (INC) o decrementa (DEC) il valore di un registro ed in molte altre circostanze. Per esempio,

se sommiamo 1Ø Hex all'accumulatore, i segnali ci diranno se dopo l'operazione il valore contenuto nell'accumulatore è nullo, se è negativo (secondo la convenzione del complemento a due) oppure se vi è stato un "overflow" o riporto.

I SEGNALI

IL SEGNALE DI RIPORTO ci dice se si è verificato un "overflow". Per esempio, se sommiamo 1Ø Hex a F3 otteniamo 1Ø3 Hex, che è troppo grande per un registro ad un singolo byte: quindi la prima cifra (1) viene eliminata, lasciando Ø 3 come risultato. Ma F3 + 1Ø non è uguale a Ø3, e perciò il segnale di riporto viene posto a uno per avvertirci che si è verificato un overflow (si è andati fuori scala). Il segnale viene posto a uno anche se si verifica un "underflow". Per esempio, se si sottrae Ø5 da Ø3 il risultato è un numero negativo espresso nella convenzione del complemento a due: possiamo quindi controllare il segnale di riporto per vedere se vi è stato underflow, in questo caso il segnale è stato posto a uno.

SEGNALE DI PARITÀ/Overflow Le modalità di utilizzo del SEGNALE DI PARITÀ/ overflow e le condizioni in cui esso viene posto a uno o azzerato sono piuttosto complicate e non hanno molta rilevanza per quanto verrà detto in questa sezione del libro.

SEGNALE DI AZZERAMENTO L'uso del segnale di azzeramento è abbastanza semplice: esso ci dice se il risultato dell'ultima operazione è stato zero. Per esempio, se prima carichiamo Ø2 Hex nel registro B e poi facciamo DEC B, il segnale di azzeramento verrà posto a zero, perché B non è diventato nullo. Se poi facciamo ancora DEC B il valore di B diventerà zero e quindi la CPU porrà a uno il segnale di azzeramento. Questo è un segnale veramente molto utile: lo vedremo ancora in seguito.

IL SEGNALE DEL SEGNO: se stiamo lavorando con la convenzione del complemento a due, questo segnale ci dice se il risultato dell'ultima operazione eseguita è stato negativo. Se era negativo secondo la convenzione del complemento a due, il segnale del segno viene posto a uno, altrimenti viene posto a zero.
Gli altri due segnali, H ed N, sono dedicati all'utilizzo esclusivo della CPU, quindi non ci preoccuperemo di esaminarli.
Si consideri il seguente esempio:

```
LD A,1Ø
LD B,1Ø
SUB A,B
```

Il numero 1Ø viene sottratto da 1Ø, con risultato zero, cosicché il segnale Z viene posto a uno; non vi è stato alcun riporto, quindi il segnale C è posto a zero; il

segnale del segno viene azzerato ad indicare che nell'accumulatore vi è un valore finale positivo:

```
Z : 1
C : Ø
S : Ø
```

Date un'occhiata ai seguenti esempi; dopo ognuno di essi vengono riportate le condizioni risultanti nei vari segnali:

```
1.                LD A,ØØ
                  LD B,B5
                  ADD A,B
   Z  :  Ø  (Zero)
   C  :  Ø  (Carry)
   S  :  Ø  (Sign)
2.                LD A,ØØ
                  SUB 2Ø
   Z  :  Ø  (Zero)
   C  :  1  (Carry)
   S  :  1  (Sign)
3.                LD A,ØØ
                  SUB B5
   Z  :  Ø  (Zero)
   C  :  1  (Carry)
   S  :  1  (Sign)
4.                LD A,ØØ
                  ADD 2Ø
   Z  :  Ø  (Zero)
   C  :  Ø  (Carry)
   S  :  Ø  (Sign)
```

Quale sarà il valore risultante dalla somma di 1Ø Hex con 7Ø Hex? Quale sarà lo stato dei segnali di azzeramento, di riporto e di segno? Dunque, aznitutto se sommiamo 7Ø Hex e 1Ø Hex otteniamo 8Ø Hex, ovvero —7F in forma di

complemento a due. Né 80 né -7F sono uguali a zero, quindi il segnale di azzeramento viene posto a zero. non c'è stato un underflow o un overflow, quindi il segnale di riporto viene posto a zero. Ma il settimo bit viene posto a uno indicando che secondo la convenzione del complemento a due il numero è negativo.

"Ma dove ci porta tutta questa storia di segnali?", vi chiederete. Usando le istruzioni JR, JP, CALL e RET nelle loro forme "condizionate", potremo effettuare "salti condizionati":

```
  JP cond¹,nnnn
CALL cond¹,nnnn
  JR cond²,dis
RET cond¹
```

cond¹ = Z/NZ/NC/C/PO/PE/M/P   dis = ± numero Hex
cond² = Z/NZ/NC/C   nnnn = indirizzo Hex a due bytes

Come potete vedere in questa figura le istruzioni di salto e di ritorno hanno come suffisso una o due lettere. Ecco cosa vogliono dire queste lettere:

Z: se il segnale di azzeramento è posto a uno.
NZ: se il segnale di azzeramento è posto a zero.
C: se il segnale di riporto è posto a uno.
NC: se il segnale di riporto è posto a zero.
PO: se la parità è dispari (cioè P/V viene posto a zero).
PE: se la parità è pari (cioè P/V viene posto a uno).
M: se negativo (cioè S viene posto a uno).
P: se positivo (cioè S viene posto a zero).

Così possiamo interpretare l'istruzione "JP Z, 705C" in questo modo: se il risultato dell'ultima operazione era zero, allora salta direttamente all'indirizzo 705C". Anche l'istruzione "CALL NC, 700A" può essere interpretata: "se l'ultima operazione non ha causato un riporto, allora passa ad eseguire la subroutine a 700A".

Per "ultima operazione" intendiamo l'esecuzione dell'ultima istruzione che abbia avuto influenza sui segnali. Istruzioni come "LD" e numerose altre non hanno alcun effetto sui segnali.

```
LD A, 10
ADD A, 05
LD B, 03
LD HL, 0735
RET Z
```

Quando, nel precedente programma si giunge a "RET Z", i segnali sono ancora fissati in modo da riflettere il valore di A dopo "ADD A, Ø5". Le varie istruzioni LD possono essere ignorate quando si considerano i segnali. Ecco una lista delle istruzioni che influenzano i segnali che ci interessano, cioè i segnali C, Z., S e P/V:

ADC
ADD
AND
BIT
CCF
CP
CPJ
CPJR
CPDR
CPL
DAA
DEC (soltanto registri da un byte)
INC (soltanto registri da un byte)
IN
INI
IND
INIR
INDR
LD A,I (nota, queste sono le uniche semplici istruzioni "LD"
LD A,R che influenzano i segnali
LDI
LDD
LDIR
LDDR
NEG
OR
OUTI
OUTD
OTIR
OTDR
POP AF (i segnali sono influenzati dal byte più elevato dello
RLA stack)

RL
RLCA
RLC
RLD
RRA
RR
RRCA
RRC
RRD
SBC
SCF
SCF
SLA
SRA
SRL
SUB
XOR

Alcune di queste istruzioni non le abbiamo ancora esaminate, perciò non iniziate a preoccuparvi. Come potete vedere la maggior parte di queste istruzioni sono relative, in un modo o nell'altro, ad operazioni matematiche. Se volete controllare esattamente quali segnali vengono modificati con ciascuna istruzione, guardate nell'Appendice C di questo libro.

Ci sono due modi molto usati di servirsi dei segnali, facendo riferimento anzitutto all'istruzione DEC, poi all'istruzione CP.

In BASIC possiamo creare un ciclo molto facilmente, mediante la sequenza FOR... TO... (STEP)... NEXT...; in linguaggio macchina le cose sono un po' diverse, anche se il principio è lo stesso. Anzitutto per formare un tipo di ciclo molto elementare, possiamo caricare in un registro il numero di volte che vogliamo far eseguire una certa operazione. Poi, al punto in cui in BASIC metteremmo il comando NEXT..., riduciamo di 1 il valore di quel registro (con DEC) e se tale valore non è ancora pari a zero facciamo eseguire di nuovo il ciclo. Guardiamo

il seguente programma per chiarire la cosa: esso stampa cinque lettere "A" usando l'istruzione "RST 10" del primo capitolo:

```
7000 0605      LD    B,05
7002 3E41      LD    A,41
7004 D7        RST   10
7005 05        DEC   B
7006 20FA      JR    NZ,7002
7008 C9        RET
```

7000: B viene fissato a 05 Hex
7001: viene caricato in A il numero 42 Hex, cioè il codice relativo alla lettera A
7004: il carattere viene stampato
7005: B viene ridotto di 1 perché viene usato come contatore
7006: se B non è arrivato a zero dobbiamo tornare a 7002 per far stampare un'altra A — usiamo l'istruzione "JR NZ", cioè 'salto relativo se non nullo'.

Il registro B viene usato molto spesso come contatore, in quanto esiste un'utilissima istruzione esclusivamente per questo registro:

"DJNZ dis", corrispondente al codice esadecimale 10XX. Questa istruzione combina "DEC B" e "JR NZ" in una sola istruzione ottenendo due vantaggi:

si usa meno memoria (soltanto un byte per l'istruzione ed uno per lo spostamento — "displacement") ed è più rapido della sequenza "DEC B", "JR NZ", il che può a volte essere importantissimo nei cicli di controllo preciso dei tempi.
Ciò vuol dire che possiamo risparmiare un byte e riscrivere così il programma:

```
7000 0605      LD    B,05
7002 3E41      LD    A,41
7004 D7        RST   10
7005 10FA      DJNZ  7001
7007 C9        RET
```

Inserite nel computer la versione modificata, ecco i dati:

```
  80 DATA "0605","3E41","D7","10
FA","C9"
```

Assicuratevi he la linea 10 sia:

```
10 LET indirizzo=28672
```

Dopo aver fatto tutto ciò e controllato che tutto sia corretto, scrivete RUN e pi PRINT AT 0,0;: RANDOMIZE USR 28672.

Ricordatevi che abbiamo bisogno di scrivere "PRINT AT 0,0;" soltanto perché altrimenti lo Spectrum ritiene di dover stampare sulla linea di Edit (o meglio subito sopra di essa), così quando ha finito ripulisce il tutto. Ora, se avete scritto tutto correttamente, troverete cinque meravigliose lettere A maiuscole scritte sul video.

Questo sistema di esecuzione dei cicli va bene quando si deve usare un conteggio semplice, ma come si fa per avere un ciclo con valori da 10 Hex a 20 Hex? In questo caso, il sistema visto ora non si può applicare perché esso si basa sul fatto che il registro contatore termina con il valore 00 Hex. Qui dobbiamo allora introdurre un'altra istruzione molto utile, CP.

Il suo significato letterale è "confronta (in inglese "compare") il seguente valore con il valore contenuto nell'accumulatore"; il risultato del confronto viene riflesso nello stato dei segnali.
Ciò che succede in pratica è che il numero che segue questa istruzione viene sottratto dal valore contenuto in A senza effettivamente caricare in A il contenuto dell'operazione. I vari segnali però vengono influenzati dall'operazione. Il valore seguente all'istruzione CP può essere un registro ad un solo byte, direttamente un numero, oppure una locazione di memoria il cui indirizzo sia contenuto in HL, IX + (dis) oppure 1Y + (dis). Ecco i codici Hex per ciascun caso di istruzione CP:

CP nn	FE nn
CP A	BF
CP B	B8
CP C	B9
CP D	BA
CPE	BB
CP H	BC
CP L	BD
CP(HL)	BE
CP(IX + (dis))	DD BE
CP(IY + (dis))	FD BE

L'istruzione di confronto può essere utile per simulare situazioni del tipo IF...THEN...; per esempio se usiamo l'istruzione "CP 20" ed il registro A contiene 20, il segnale Z verrà posto a 1 (20-20=0), se A è minore di 20 verrà posto a 1 il segnale C, e se è maggiore di 20, il segnale C verrà posto a zero.

Le seguenti situazioni possono essere interpretate con significato quasi analogo a quello del BASIC:

1. CP 5F
   JP Z, 7000

Come potete vedere il valore 5F viene sottratto teoricamente da A. In questo caso, se il risultato del calcolo è nullo, si effettuerà un salto. L'unico modo in cui il risultato potrà essere è che l'accumulatore contenga 5F prima del calcolo (5F-5F=00), quindi potremmo dire:

If A=5F (Hex)THEN GOTO 7000(Hex)

2. CP 2C
   JP C, 7000

In quest'altro caso si sottrae teoricamente 2C da A. (Ricordate che diciamo teoricamente perché il risultato non viene mai messo in A). Ora, se A è maggiore di 2C, si ha un risultato negativo, ovvero un "underflow", ed il segnale di riporto viene posto ad uno. Potremo scrivere allora così:

IF A< 2C(Hex)THEN GOTO 7000(Hex)

3. CP 10
   JP NC, 7000

Nel terzo caso, se da A togliamo 10 e A conteneva un numero maggiore di 10, il risultato è positivo; quindi non ci sarà overflow e il segnale di riporto viene posto a 0. Potremmo scrivere:

```
IF A>10(Hex)THEN GO TO 7000(Hex)
```

Tornando ora al nostro problema di costruire un ciclo che non termini a zero, consideriamo il segnale programma che ha un ciclo da 10 Hex a 1F Hex:

```
7000 3E10     LD    A,10
7002 3C       INC   A
7003 FE20     CP    20
7005 20FB     JR    NZ,7002
7007 C9       RET
```

Anzitutto si carica 10 in A, il valore da cui deve iniziare il ciclo. Poi, all'indirizzo 7002 l'accumulatore viene aumentato di 1 e con "CP 20" lo si confronta con 20: se l'accumulatore non contiene 20, il segnale Z viene posto a zero e quindi si ritorna a 7002. Facciamo il confronto con 20 perché questo è il numero seguente all'ultimo numero con cui vogliamo far eseguire il ciclo: se facessimo il confronto con 1F, si incrementerebbe A fino a 1E, eseguendo il ciclo, lo si porterebbe poi a 1F ma si avrebbe subito il ritorno senza esecuzione del ciclo.

Espandendo un po' questo programma, eccone uno che stampa 16 lettere A in colonna, dalla 16 alla 31 decimale, ovvero da 10 a 1F Hex:

```
7000 0610        LD   B,10
7002 3E17        LD   A,17
7004 D7          RST  10
7005 78          LD   A,B
7006 D7          RST  10
7007 3E41        LD   A,41
7009 D7          RST  10
700A 78          LD   A,B
700B FE20        CP   20
700D 20F3        JR   NZ,7002
700F C9          RET
```

7000: B viene posto a 10 invece di A, poiché avremo bisogno di A per la stampa
7002: In A viene caricato 17 Hex, ovvero il codice di controllo per il TAB
7004: Il controllo TAB viene "inviato"
7005: Ora carichiamo nell'accumulatore il numero di colonna, contenuto in B
7006: Il numero di colonna viene "inviato"
7007: In A viene caricato il codice per la lettera "A"
7009: La lettera viene stampata
700ACarichiamo in A il valore del contatore in B, per poter eseguire il test
700B: Si fa il confronto con 20 perché l'ultimo valore da usare è 1F (31 decimale)
700D: Se l'accumulatore non è ancora giunto a 20, si torna a stampare un'altra lettera "A"
700F: Ritorno al BASIC

Le cose sono abbastanza semplici con i cicli che fanno uso di registri ad un solo byte, ma quando c'è bisogno di contare con due byte si incontra un'ulteriore complicazione. Decrementando di 1 un registro a due byte si influenzano anche i segnali e quindi non funzionerà il controllo di fine ciclo. In altre parole non si può fare così, per esempio:

```
LD BC, 062D
DEC BC
JR NZ....
```

Discuteremo di come superare questo problema nel quarto capitolo, dove impareremo tutto ciò che riguarda le istruzioni che operano "logicamente" su un registro.

L'INSIEME DEI CARATTERI

Prima di presentarvi una piccola routine molto simpatica che permette di avere caratteri ad altezza doppia, facciamo un ripasso dell'insieme dei caratteri.
È molto conveniente che nello ZX Spectrum esistano i simboli grafici definiti dall'utente, dato che ciò significa che non è necessario che io dia tante spiegazioni! Dovreste già sapere infatti che ogni carattere è formato da 64 punti in un quadrato di otto per otto. Molti di voi sanno anche che ogni "fetta" di otto punti viene memorizzata come otto bit, ovvero un byte.

128	64	32	16	8	4	2	1	
0	0	0	0	0	0	0	0	00
0	0	1	1	1	1	0	0	3C
0	1	0	0	0	0	1	0	42
0	1	0	0	0	0	1	0	42
0	1	1	1	1	1	1	0	7E
0	1	0	0	0	0	1	0	42
0	1	0	0	0	0	1	0	42
0	0	0	0	0	0	0	0	00

L'insieme usuale di caratteri Sinclair è in memoria dalla locazione 3000 in poi: qui sono definiti tutti i caratteri, dallo spazio al simbolo di copyright. I simboli grafici definiti dall'utente sono invece memorizzati separatamente e i simboli vengono calcolati dalla ROM. Se ridefiniamo altrove l'insieme dei caratteri dobbiamo cambiare il valore in CHARS (5CB6) fra le variabili di sistema. Questo valore di solito è 3C00, cioè 100 Hex in meno dell'indirizzo dell'insieme di caratteri; dovete ricordarvi che quando definite un nuovo insieme di caratteri, lo si deve cambiare a 100 Hex in meno dell'effettivo indirizzo, cosicché potrete usare il nuovo insieme.

CARATTERI A DOPPIA ALTEZZA

Per creare un insieme di caratteri a doppia altezza dobbiamo in effetti creare due nuovi insiemi, uno per la metà superiore del carattere, l'altro per la metà inferiore. Per generare questi due insiemi di caratteri, prendiamo un carattere normale e lo "allunghiamo" su due mezzi caratteri (vedi l'esempio con la letetra A).
Quindi i quattro byte superiori li ricopiamo ciascuno due volte nel primo insieme di caratteri, quello per la metà superiore. Poi dobbiamo copiare i quattro byte inferiori, ciascuno due volte, nel secondo insieme di caratteri per la metà inferiore.

Usando il linguaggio macchina possiamo ottenere che questo lavoro complicato venga svolto in un batter d'occhio. Anzitutto dobbiamo avere un ciclo esterno che passa per tutti i 96 caratteri. Poi ci vuole un ciclo interno per la metà superiore di ogni carattere e un altro ciclo per quella inferiore.

Ciclo DO per ciascun carattere ←
Ciclo DO per la metà superiore ←
Ciclo DO per la metà inferiore ←
NEXT = carattere successivo

Ora dobbiamo decidere dove mettere questi insiemi di caratteri. Il posto migliore è al di sopra del RAM TOP, e faremo proprio così. Se avete un disassembler, un editor o altro programma per il linguaggio macchina che risiede sopra il RAM TOP, dovrete cambiare gli indirizzi. Dove c'è scritto PUT1 e PUT2 nella prima parte del programma usate i seguenti indirizzi.

48k : (PUT1);FA(PUT2);FD
16K ; (PUT1):7A(PUT2);7D

Il nostro primo compito è quello di preparare un ciclo che possa elaborare ciascun carattere. Avremo bisogno di tenere a disposizione due indirizzi, che ci dicano dove ci troviamo nei due diversi insiemi di caratteri.

```
LD HL,BOTSET   21 00 [PUT2]
PUSH HL
LD HL,TOPSET   21 00 [PUT1]
LD DE,SNCSET   11 00 3D
LD C,60        0E 60
```

Così questo è il nostro primo pezzo. Fissiamo l'indirizzo per l'insieme della metà inferiore e lo mettiamo in uno stack. Poi fissiamo l'indirizzo per la metà superiore e lo lasciamo in HL. Nel registro DE carichiamo SNCSET (3D00), che è l'inizio dell'insieme di caratteri Sinclair. Infine, in C carichiamo 60 Hex perché ci sono 60 Hex (96 decimale) caratteri da considerare. Ora ci vuole il ciclo per la metà superiore dei caratteri.

```
TPHALF   LD B,04        06 04
TSLICE   LD A,(DE)      1A
         LD (HL),A      77
         INC HL         23
         LD (HL),A      77
         INC HL         23
         INC DE         13
         DJNZ TSLICE    10 F8
```

Si noti che la prima istruzione carica 4 in B: in questo modo si usa B come contatore, e poiché dobbiamo elaborare quattro "fette", dobbiamo caricare il numero 4. Nell'accumulatore viene poi caricato il byte su cui è puntato DE: ricordate che DE punta all'insieme di cartteri Sinclair. Questo byte viene trasferito nell'insieme delle metà superiori dei caratteri. Poi HL viene incrementato ed il byte viene di nuovo memormizzato in (HL): come vi ricordate dobbiamo mettere due volte ciascun byte, per avere l'effetto di allungamento. Infine DE viene incrementato così da puntare alla successiva "fetta" del carattere.

Questa procedura viene ripetuta quattro volte usando l'istruzione DJMZ che abbiamo visto prima.
L'ultimo passo, prima di chiudere il ciclo esterno è preparato le metà inferiori dei caratteri. Ciò viene eseguito con una procedura molto analoga a quella per le metà superiori che abbiamo visto prima.

```
        EX HL,(SP)    E3
        LD B,04       06 04
BSLICE  LD A,(DE)     1A
        LD (HL),A     77
        INC HL        23
        LD (HL),A     77
        INC HL        23
        INC DE        13
        DJNZ BSLICE   1 F8
        EX HL,(SP)    E3
        DEC C         0D
        JR NZ,TPHALF  20 E6
        POP HL        E1
        RET           C9
```

Questo pezzo di programma inizia in modo un po' diverso, con l'istruzione EX HL, (SP). Per coloro che non la conoscono, si tratta di un'istruzione molto utile che semplicemente scambia il valore in HL con il valore in cima allo stack. Qui essa viene usata in modo che HL passi a puntare dall'insieme delle metà superiori dei caratteri a quello delle metà inferiori; così l'insieme dei caratteri inferiori viene memorizzato ad un indirizzo diverso, altrimenti si avrebbe un sacco di confusione! Dopo "DJNZ BSLICE" c'è un'altra istruzione "EX HL, (SP)", così si scambiano di nuovi i valori e HL è pronto per l'insieme superiore dei caratteri. Il registro C viene decrementato e se non è pari a zero si esegue un salto relativo per passare

```
7000  21007D   LD    HL,7D00
7003  E5       PUSH  HL
7004  21007A   LD    HL,7A00
7007  11003D   LD    DE,3D00
700A  0E60     LD    C,60
700C  0604     LD    B,04
700E  1A       LD    A,(DE)
700F  77       LD    (HL),A
7010  23       INC   HL
7011  77       LD    (HL),A
7012  23       INC   HL
7013  13       INC   DE
7014  10F8     DJNZ  700E
7016  E3       EX    (SP),HL
7017  0604     LD    B,04
7019  1A       LD    A,(DE)
701A  77       LD    (HL),A
701B  23       INC   HL
701C  77       LD    (HL),A
701D  23       INC   HL
701E  13       INC   DE
701F  10F8     DJNZ  7019
7021  E3       EX    (SP),HL
7022  0D       DEC   C
7023  20E7     JR    NZ,700C
7025  E1       POP   HL
7026  C9       RET
```

al carattere successivo. Se non ci sono più altri caratteri, vengono eseguite le ultime due istruzioni, "POP HL" e "RET". L'istruzione "POP HL" serve per rimuovere dallo stack il valore più elevato, cioè il puntatore all'insieme dei caratteri.

Per far funzionare questa routine inserite i codici esadecimali come istruzioni DATA alla fine del programma Hex Loader. Ricordatevi di scegliere come indirizzi per il caricamento dei valori appropriati per le versioni a 16K o a 48K, a seconda della macchina che avete. Qui sono indicati i valori per la versione a 16K, ma se gli utenti con la versione a 48K guardano indietro nel libro, troveranno il punto dove ho spiegato come fare per modificarli. Se state usando un monitor in linguaggio macchina che si serve degli indirizzi, da 7ACO e 7FFF sulla versione a 16K, oppure da FA00 a FFFF su quella a 48K, si dovranno fare alcune piccole modifiche agli indirizzi, altrimenti il programma scriverà i due insiemi di caratteri al posto del monitor!

Vediamo ora come usare questo generatore di caratteri a doppia altezza. Supponiamo che abbiate già scritto le istruzioni DATA e fatto eseguire il loader; adesso volete sapere come usare le nuove lettere.

Anzitutto fissiamo i seguenti valori per queste due variabili:

Per la versione a 16K — LET TOP = 121
LET BOT = 124

Per la versione a 48K — LET TOP = 249
LET BOT = 252

Supponiamo ora di voler stampare "Hello":

```
10 POKE 23607, TOP: PRINT "Hello"
20 POKE 23607, BOT: PRINT "Hello": POKE 23607, 60
```

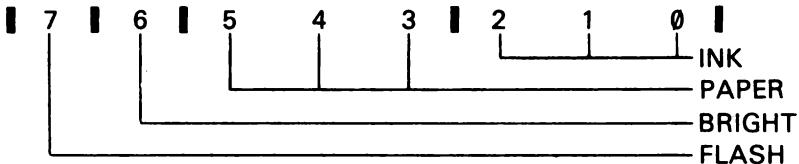
Nella prima linea puntiamo CHARS (una variabile di sistema, vedi Appendice C) al nostro insieme delle metà superiori dei caratteri, in modo da poter scrivere le metà superiori. Poi, per poter aggiungere subito sotto le metà inferiori usiamo di nuovo il comando POKE per puntare CHARS all'insieme delle metà inferiori. L'ultimo comando POKE alla fine della linea 20 serve per riportare la stampa al modo normale.
Provate a vedere cosa succede se si da in modo diretto il comando

"POKE 2360 7, TOP".
Provate anche
"POKE 23607,0": perché fa così?
Riuscite a trovare un buon uso per questo fatto?

CAPITOLO 3

I BIT

Come già sapete, un byte è costituito da otto bit. Ciascuno di questi bit è una cifra binaria ("bit" è l'abbreviazione di "binary digit") che rappresenta un numero se posto pari a uno. Tutti questi insieme formano un byte. A volte usiamo un byte non tutto insieme come una "parola da otto bit", ma come bit separati che rappresentano segnali oppure numeri più piccoli. Abbiamo già visto il registro F ed i suoi bit, e come ciascun bit ci dice qualcosa riguardo l'ultima operazione o confronto eseguito. Il file di attributi di colore (che inizia a 5800 Hex) ha un byte per ciascun quadrato di carattere sul video; ogni byte di attributi comunica al microprocessore il colore di inchiostro "INK" il colore della pagina "PAPER" e l'inserimento o meno della stampa lampeggiante o brillante.

Ecco come è organizzato un byte di attributi:

Se vogliamo far lampeggiare un carattere poniamo a uno il bit numero sette, ma nessun altro bit deve essere modificato. Per fare ciò usiamo l'istruzione "SET".

Il seguente programma fa lampeggiare il primo carattere del video.

```
7000 210058   LD   HL,5800
7003 CBFE     SET  7,(HL)
7005 C9       RET
```

7000: In HL viene caricato 5800 Hex. Questo è l'indirizzo del primo byte di attributi, che si riferisce al primo carattere sul video.
7003: L'istruzione SET pone a uno il bit numero 7 della locazione (HL), e

poiché HL è puntato al primo attributo e il bit numero 7 è quello relativo al lampeggiamento, si ha che il primo carattere si mette a lampeggiare.

7005: Ritorno al BASIC.

L'istruzione SET pone a uno soltanto il bit specificato lasciando inalterati tutti gli altri bit. Può essere usata per modificare un qualsiasi bit (da zero a sette) di ogni registro a byte singolo (A, B, C, D, E, H o L) o qualsiasi locazione puntata da (HL), (IX + dis) o (IY + dis). Ecco la lista di tutte le istruzioni SET (dove (X) è un qualunque numero da zero a sette):

SET (x), A
SET (x), B
SET (x), C
SET (x), D
SET (x), E
SET (x), H
SET (x), L
SET (x), (HL)
SET (x), (IY + dis)
SET (x), (IX + dis)

Complementare all'istruzione SET è RES: essa funziona in modo simile, tranne che pone a zero il bit appropriato.

RES (x), A
RES (x), B
RES (x), C
RES (x), D
RES (x), E
RES (x), H
RES (x), L
RES (x), (HL)
RES (x), (IY + dis)
RES (x), (IX + dis)

Così se vogliamo far sì che il primo carattere non sia più lampeggiante possiamo azzerare il settimo bit usando l'istruzione RES.

```
7000 210058     LD    HL,5800
7003 CBBE       RES   7,(HL)
7005 C9         RET
```

Poiché esistono molte diverse istruzioni SET e RES, sarebbe uno spreco di spazio riportarne qui la lista completa: la potrete trovare tutte nell'Appendice A di questo libro oppure del vostro manuale Sinclair.

Il seguente programma mostra il funzionamento dell'istruzione SET, facendo lampeggiare le prime otto righe dello schermo.

```
7000 210058   LD   HL,5800
7003 0600     LD   B,00
7005 CBFE     SET  7,(HL)
7007 23       INC  HL
7008 10FB     DJNZ 7005
700A C9       RET
```

7000: In HL viene caricato l'indirizzo del primo byte, 5800.
7003: B, che viene usato come contatore, viene fissato a 00, cosicché vengono elaborati i primi 256 caratteri.
7005: Il bit del lampeggiamento viene posto a uno nella locazione (HL).
7007: HL viene spostato in avanti all'attributo successivo.
7008: L'istruzione DJNZ fa ripetere l'operazione per 256 volte.
700A: Ritorno al BASIC.

Proviamo questo programma: usando un programma BASIC di caricamento, cambiate la linea 80 con:

```
80 DATA "210058","0600","CBFE"
,"23","10FB","C9"
```

Controllate che la linea 10 sia:

```
10 LET indirizzo=28672
```

Poi scrivete
RUN
e quando ricevete il messaggio di errore scrivete
RANDOMIZE USR 28672

Come vedete, le prime otto righe dello schermo sono lampeggianti. Se non è così avete sbagliato a scrivere qualcosa. Se avete perso il loader, ricaricatelo e provate di nuovo, altrimenti controllate la linea 80. Ora, giusto per dimostrarvi che con questa operazione non si è modificato nient'altro in quei quadrati di carattere, scrivete:

LIST: RANDOMIZE USR 28672

e voilà! ecco un programma che lampeggia nelle sue prime righe!! Proviamo ora a migliorare l'aspetto di questa pagina, facendo stampare i caratteri in modo brillante, con una piccola modifica.

```
7000 210058    LD    HL,5800
7003 0600      LD    B,00
7005 CBF6      SET   6,(HL)
7007 23        INC   HL
7008 10FB      DJNZ  7005
700A C9        RET
```

Invece di porre a uno il settimo bit, poniamo a uno il sesto, cioè il bit dell'alta luminosità. Quindi nella linea 80 cambiate "CBFE" con "CBFG". Scrivete
RUN

e poi:
LIST: RANDOMIZE USR 28672.

Ecco che, se il colore della pagina che state usando è il bianco, avrete un bianco che più bianco non si può!

Esiste un'altra istruzione relativa ai bit che dobbiamo esaminare:
"BIT".

Questa istruzione esegue un test su un certo bit e se quel bit è nullo, il segnale di azzeramento viene posto a uno, se invece esso è uno il segnale Z viene posto a zero. Le diverse forme dell'istruzione BIT sono analoghe a quelle di SET e di RES, e come quelle, anche questa istruzione richiede il prefisso CB. I codici possono essere trovati nell'Appendice A di questo libro o nel manuale Sinclair.

BIT (X), A
BIT (X), B
BIT (x), C
BIT (x), D
BIT (x), E
BIT (x), H
BIT (x), L
BIT (x), (HL)
BIT (x), (IX + dis)
BIT (x), (IY + dis)

Il seguente programma fa passare tutti i caratteri da lampeggianti a fissi e viceversa.

```
7000 210058   LD   HL,5800
7003 CB7E     BIT  7,(HL)
7005 2804     JR   Z,700B
7007 CBBE     RES  7,(HL)
7009 1802     JR   700D
700B CBFE     SET  7,(HL)
700D 23       INC  HL
700E 7C       LD   A,H
700F FE5B     CP   5B
7011 20F0     JR   NZ,7003
7013 C9       RET
```

7000: In HL viene caricato l'indirizzo iniziale del file degli attributi.
7003: Il bit per il lampeggiamento di (HL) viene sottoposto a test.
7005: Se il carattere non lampeggia si passa a 700B.
7007: Altrimenti se ne arresta il lampeggiamento.
7009: Salto relativo a 700D.
700B: Si fa lampeggiare il carattere.
700D: Si sposta HL al successivo attributo.
700E: Si porta H in A e si fa un test per vedere se si è arrivati alla fine del file degli attributi.
7011: Se non si è alla fine si salta a 7003 per elaborare il byte successivo.
7013: Ritorno al BASIC.

Proviamo. Cambiate la linea 80 del loader con:

```
 80 DATA "210058","CB7E","2804"
,"CBBE","1802","CBFE","23","7C",
"FE5B","20F0","C9"
```

Poi scrivete

RUN
e
RANDOMIZE USR 28672

Lo schermo lampeggia!

Ora scrivete di nuovo

RANDOMIZE USR 28672

e lo schermo torna fisso. Provate a scrivere:

PRINT AT 10,0; FLASH 1; "Controllo routine FLASH 2"

Così possiamo far lampeggiare tutto tranne il testo del messaggio:

RANDOMIZE USR 28672

Cerchiamo ora di raccogliere tutto ciò che abbiamo imparato e ci cimentiamo con un nuovo programma.

TYPEWRITER

Il programma che scriveremo si chiama "Typewriter" (cioè macchina da scrivere). È un programma molto semplice che consente di usare lo schermo come una pagina su cui scrivere con una normale macchina: lo potete usare per preparare la stampa delle istruzioni di un programma, se avete una stampante, oppure giusto per divertirvi se non l'avete. Le principali funzioni di "Typewriter" sono:

a) controlli del cursore: su, giù, sinistra e destra;
b) ripetizione automatica;
c) ritorno del cursore alla posizione "Home" (in alto a sinistra);
d) funzione di cancellazione;
e) tasto di ritorno carrello ovvero di immissione.

Anzitutto vogliamo avere due byte di dati che ci dicano in quale punto del video si trova il cursore, e due byte di dati che ci dicono dove scrivere nel file degli attributi. Un quinto byte di dati ci dice quale tasto è stato premuto. Per designare questi byte di dati useremo i seguenti nomi:

PRINT — 6D00
ATTRB — 6D02
KEY — 6D04

Inizieremo quindi a scrivere il programma dalla locazione 6D05, subito dopo questi dati.

Finora non ho ancora spiegato come si possa avere in linguaggio macchina la segnalazione circa la avvenuta pressione di un tasto.
La cosa è molto semplice: fra le variabili di sistema c'è una locazione chiamata 'LASTK", che contiene il codice dell'ultimo tasto che è stato premuto. Essa viene aggiornata ogni 50-esimo di secondo (nelle macchine funzionanti in Europa), oppure ogni 60-esimo di secondo (negli U.S.A.). La procedura di attesa della pressione di un tasto e di immissione nell'accumulatore del codice del tasto premuto è la seguente:

1. azzerare LAST K;
2. caricare LAST K;
3. se l*accumulatore è zero, tornare al passo 2.

L'aspetto positivo di questa procedura è che il sistema genera la ripetizione automatica della procedura stessa.

Possiamo servirci dell'istruzione RST 10 per scrivere sul video usando il codice di controllo AT, però il cursore lampeggiante non può essere ottenuto con RST 10 perché così si cancellerebbe il carattere sotto il cursore. Ciò dipende dal fatto che il nostro cursore è diverso da quello dello Spectrum: invece di andare in mezzo ai caratteri, esso sarà sopra i caratteri stessi, come quadratino lampeggiante. Così ci risparmiamo la fatica di spostare tutto in avanti o all'indietro ogni volta che il cursore viene spostato.

Quando viene premuto un tasto il programma esegue una di due possibili operazioni: stampare il carattere oppure compiere un'operazione con il cursore, per esempio una cancellazione o spostare il cursore verso l'alto. Si deve quindi far conoscere al programma quali sono i codici da stampare e quali quelli relativi a controlli del cursore. Facciamo in modo che il programma consulti una tabella di codici, ciascuna dei quali viene seguito da un indirizzo a due byte: se il codice del tasto premuto coincide con uno dei codici della tabella, il programma salta all'indirizzo relativo a quel codice, altrimenti il codice viene interpretato come un carattere e viene stampato. Ecco la tabella:

```
DEFB 07    — EDIT              0705 6D
DEFB 08    — Sinistra          0883 6D
DEFB 09    — Destra            095A 6D
DEFB 0A    — Giù               0A69 6D
DEFB 0B    — Su                0B76 6D
DEFB 0C    — Cancella          0C94 6D
DEFB 0D    — Immissione        0DA6 6D
DEFB 0F    — Simboli grafici   0FA5 6D
DEFB FF    — Fine tabella      FF
```

Nella tabella ci sono i codici di un certo numero di tasti di controllo; ciascun codice viene seguito da un numero a due byte, che punta all'inizio della routine che svolge l'operazione relativa a quel tasto:

07	—	EDIT	—	Muovere il cursore alla posizione 0,0.
08	—	SINISTRA	—	Muovere il cursore a sinistra.
09	—	DESTRA	—	Muovere il cursore a destra
0A	—	GI7	—	Muovere il cursore verso il basso.
0B	—	S7	—	Muovere il cursore verso l'alto.

0C	—	DELETE	—	Cancellare il carattere.
0D	—	ENTER	—	Ritorno carrello.
0F	—	GRAPHICS	—	Uscita dal programma "Typewriter".

La funzione DELETE cancella il carattere sotto il cursore e sposta il cursore di una posizione verso sinistra. ENTER muove il cursore all'inizio della linea successiva. GRAPHICS fa terminare l'esecuzione del programma, tornando al BASIC e lasciando lo schermo inalterato e pronto per effettuare una copia con COPY.

Ora che abbiamo chiarito alcuni problemi possiamo disegnarci un diagramma di flusso. Nella pagina seguente troverete il diagramma completo per il programma "Typewriter".

Partendo da questo diagramma di flusso possiamo scrivere il nostro programma con sistematicità, scrivendo il pezzo di programma relativo ad ogni singola casella. Cominciamo dalla prima.

INIZIO: PORRE PRINT ALLA POSIZIONE 0,0

Come abbiamo deciso, la posizione del cursore, indicata come coordinate per il comando AT, viene memorizzata nei byte di dati denominati PRINT (indirizzo 6D00 Hex). Quindi, molto semplicemente, per la prima casella basta caricare 0,0 sia in PRINT (numero di colonna) che in PRINT+1 (numero di riga).

```
6D05 RESET  LD HL,0000                21 00 00
            LD (PRINT), HL            22 00 6D
```

TROVARE LA POSIZIONE DI STAMPA PER GLI ATTRIBUTI

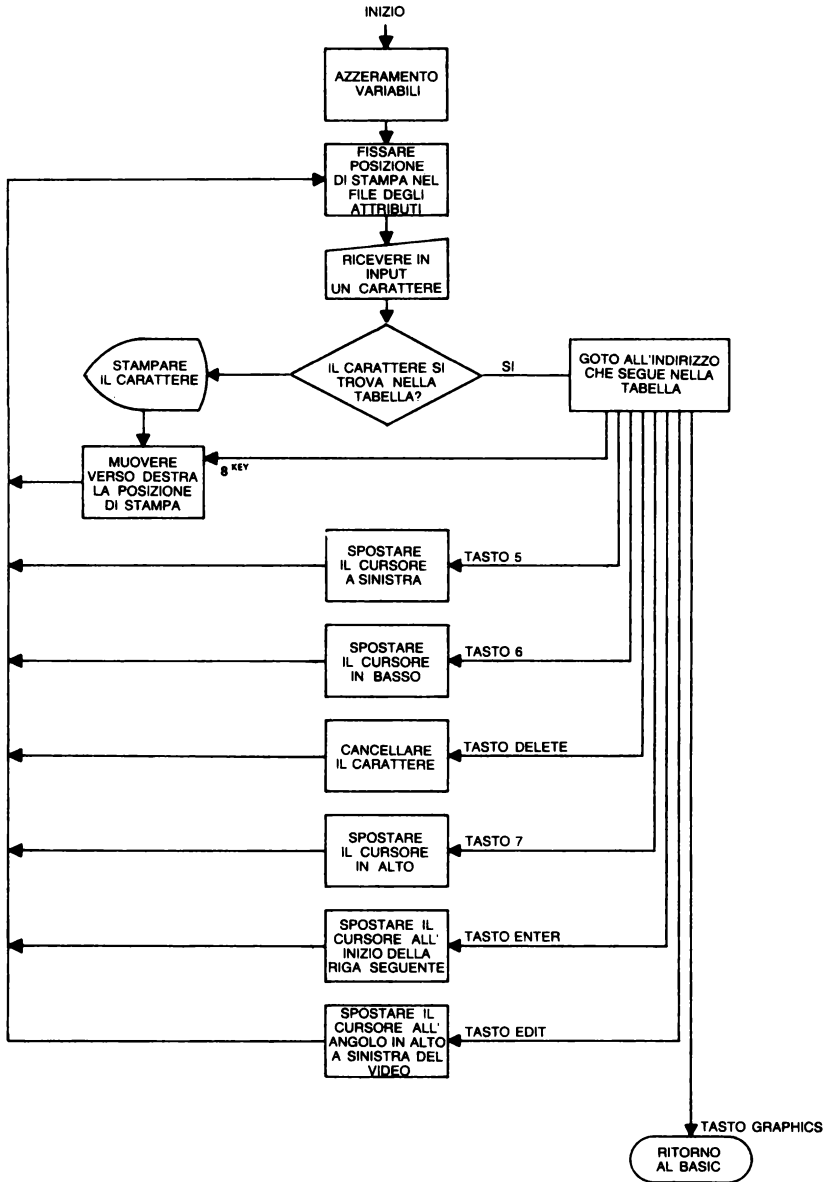



DIAGRAMMA DI FLUSSO PER IL PROGRAMMA "TYPEWRITER"

Ciò richiede che si trovi l'indirizzo del byte corrispondente nel file degli attributi una volta che siano note le coordinate AT. Per far ciò moltiplichiamo il numero di linea per 32, aggiungiamo il numero di colonna e sommiamo poi il risultato al valore di ATTRB, cioè l'inizio del file degli attributi.

```
6D0B ATRSET    LD DE, (PRINT)   ED 5B 00 6D
               LD C,E           4B
               LD E,D           5A
               LD D,00          16 00
               LD HL,ATTRBS     21  02  6D
               LD B,20          06 20
MULT 32        ADD HL,DE        19
               DJNZ MULT32      10 FD
               LD E,C           59
               ADD HL,DE        19
               LD(ATTRB),HL     22 02 6D
```

L'ultima istruzione si riferisce all'indirizzo 6D02, che come abbiamo detto è ATTRB, cioè la posizione di stampa nel file degli attributi. Non è ancora necessario immettere nel computer queste istruzioni: cercate soltanto di capire un pezzo di programma alla volta e poi alla fine ci occuperemo di come immettere e fare girare il programma completo.

POSIZIONARE IL CURSORE SUL VIDEO

Ciò è veramente molto semplice. HL contiene la posizione di stampa nel file degli attributi, dato che era stata calcolata nel precedente pezzo del programma. È sufficiente quindi porre a 1 il bit numero 7, il bit del lampeggiamento nella locazione (HL).

```
6D20   CURPUT     SET7,(HL)    CB  FE
```

ATTENDERE LA PRESSIONE DI UN TASTO

Come già abbiamo descritto prima, si azzera la variabile LASTK (indirizzo 5C08) e si rimane in attesa che cambi.

```
6D22  KEYGET    LD A,00          3E 00
                LD HL,LAST K     21 08 5C
                LD(HL),00        36 00
      KEYTST    CP(HL)           BE
```

```
        JR Z,KEYTST          28 FD
        LD A,(HL)            7E
        LD (KEY),A           32 Ø4 6D
```

Il codice del tasto che è stato premuto viene memorizzato nella variabie KEY (indirizzo 6DØ4).

TOGLIERE IL CURSORE

A questo punto togliamo semplicemente il cursore dal video usando l'istruzione RES:

```
6D3Ø NOCURS     LD HL,(ATTRB)      2A Ø2 6D
                RES 7,(HL)         CB BE
```

KEY SI TROVA NELLA TABELLA?

Dobbiamo ora andare a vedere se è stato premuto un tasto speciale, come DELETE o ENTER, oppure se è un codice che deve solo essere stampato. A questo fine useremo la tabella che abbiamo visto sopra, la quale si trova memorizzata dalla locazione TABLE in poi. La procedura sarà come segue:

1. si punta DE all'inizio della tabella, cioè a TABLE (indirizzo 6DAC);
2. si vede se il contenuto della locazione (DE), ossia il primo codice nella tabella, è uguale al codice del tasto premuto, variabile KEY;
3. in caso affermativo si sposta DE in avanti di un byte;
4. si leggono i due byte successivi nella tabella e li si mette in HL, poi si salta con JP (HL) alla routine relativa a quel codice;
5. se invece il codice nella tabella non corrisponde al codice in KEY, si sposta DE in avanti di tre byte, cioè fino al successivo codice nella tabella;
6. se questo codice è FF, si può procedere a stampare sul video il codice in KEY, altrimenti si torna al passo due.

```
6D35  SEARCH    LD DE,TABLE        11 AC 6D
                LD HL, KEY         21 Ø4 6D
      TSTKEY    LD A,(DE)          1A
                CP(HL)             BE
                JR NZ,NEXBYT       2Ø Ø7
                INC DE             13
                EX DE, HL          EB
                LD E,(HL)          5E
```

```
          INC HL            23
          LD D,(HL)         56
          EX DE,HL          EB
          JP (HL)           E9
NEXBYT    INC DE            13
          INC DE            13
          INC DE            13
          CP FF             FE FF
          JR NZ,TSTKEY      20 EE
```

Ci occuperemo più avanti delle parti di programma relative ai tasti speciali.

STAMPARE UN CARATTERE

Abbiamo visto nel primo capitolo l'istruzione RST 10 e come essa permetta di stampare con molta facilità sul video. Useremo il codice di controllo 16 e RST 10 per mettere sul video il carattere relativo al tasto premuto.

```
6D4D PUT KEY    LD DE, (PRINT)     ED 5B 00 6D
                LD A,16            3E 16
                RST 10             D7
                LD A,D             7A
                RST 10             D7
                LD A, E            7B
                RST 10             D7
                LD A, (HL)         7E
                RST 10             D7
```

SPOSTARE IL CURSORE A VERSO DESTRA

Dopo che si è stampato un carattere, il cursore deve essere spostato verso destra in una posizione. Se prima di essere spostato si trova nella colonna 32, dovrà essere spostato alla prima colonna della linea successiva, caricando il valore 00 nella locazione 6D00 (numero di colonna) e poi saltando alla sezione DOWN in cui il cursore viene spostato verso il basso. Questa procedura viene eseguita quando si preme il tasto "cursore a destra". Compiuta l'operazione, si passa alla sezione ATRSET, quella vista per seconda.

```
6D5A RIGHT      LD HL,6DØØ         21 ØØ 6D
                INC (HL)           34
                LD A, 2Ø           3E 2Ø
                CP (HL)            BE
                JR NZ, ATRSET      2Ø A8
6D63 NEXLIN     LD (HL),ØØ         36 ØØ
```

SPOSTARE IL CURSORE VERSO IL BASSO

Il cursore viene spostato verso il basso di una posizione, a meno che non si trovi già sulla 22-esima linea del video. Dopodiché si passa alla sezione ATRSET.

```
6D69 DOWN       LD HL,6DØ1         21 Ø1 6Ø
                LD A,15            3E 15
                CP (HL)            BE
                JP Z,ATRSET        CA ØB 6D
                INC (HL)           34
                JP ATRSET          C3 ØB 6D
```

SPOSTARE IL CURSORE VERSO L'ALTO

Il cursore viene spostato verso l'alto di una posizione, a meno che non si trovi già sulla prima linea del video. Poi si salta alla sezione ATRSET.

```
6D76 CUR UP     LD HL,6DØ1         21 Ø1 6D
                LD A,ØØ            3E ØØ
                CP (HL)            BE
                JP Z, ATRSET       CA ØB 6D
                DEC (HL)           35
                JP ATRSET          C3 ØB 6D
```

CURSORE VERSO SINISTRA

Il cursore viene spostato a sinistra, a meno che non si trovi sulla colonna zero, nel qual caso si carica nella locazione 6DØØ (numero di colonna) il valore 1F Hex, ovvero 31 decimale, che indica l'ultima colonna, e poi si salta alla sezione CUR UP. Se si riesce a spsotare il cursore verso sinistra, si passa poi alla sezione ATRSET.

```
6D 83 LEFT      LD HL,6DØØ        21 ØØ 6D
                DEC (HL)          35
                LD A,FF           3E FF
                CP (HL)           BE
                JP NZ, ATRSET     C2 ØB 6D
                LD (HL) 1F        36 1F
                JP CUR UP         C3 76 6D
```

STAMPARE UNO SPAZIO

Questa è la routine di cancellazione. Usando l'istruzione RST 1Ø si stampa uno spazio alle coordinate correnti, poi si riporta indietro verso sinistra saltando alla sezione LEFT.

```
6D94 DELETE     LD DE,(6DØØ)      ED 5B ØØ 6D
                LD A,16           3E 16
                RST 1Ø            D7
                LD A, D           7A
                RST 1Ø            D7
                LD A, E           7B
                RST 1Ø            D7
                LD A,2Ø           3E 2Ø
                RST 1Ø            D7
                JP LEFT           C3 83 6D
```

RITORNO AL BASIC

Quando si preme il tasto "shift-9", ovvero il tasto GRAPHICS, si può ritornare al BASIC.

```
6DA5  EXIT RET C9
```

METTERE IL CURSORE AL MARGINE SINISTRO

In effetti non è questa l'operazione che qui si esegue, dato che essa viene fatta eseguire dall'ultima linea dell'istruzione RIGHT, chiamata NEXLIN. Questa sezione viene richiamata quando si preme il tasto Enter.

```
6DA6 ENTER      LD HL,6DØØ        21 ØØ 6D
                JP NEXLIN         C3 63 6D
```

LA TABELLA DEI CODICI

Questo è l'ultimo pezzo di listato prima che possiamo iniziare ad immettere nel

computer il programma completo. Si tratta della tabella degli indirizzi per i salti alle routine relative ai tasti speciali.

```
6DAC TABLE    DEFB Ø7 'EDIT'       Ø7
              DEFB Ø56D            Ø5 6D
              DEFB Ø8 'LEFT'       Ø8
              DEFB 836D            83 6D
              DEFB Ø9 'RIGHT'      Ø9
              DEFB 5A6D            5A 6D
              DEFB ØA 'DOWN'       ØA
              DEFB 696D            69 6D
              DEFB ØB 'UP'         ØB
              DEFB 766D            76 6D
              DEFB ØC 'DELETE'     ØC
              DEFB 946D            94 6D
              DEFB ØD 'ENTER'      ØD
              DEFB A66D            A6 6D
              DEFB ØF 'GRAPHICS'   ØF
              DEFB FF 'END'        FF
```

Ora che abbiamo esaminato tutto il programma nelle sue varie sezioni, eccone un listato completo, che potrete usare come punto di riferimento.

```
6D05 21 00 00      LD   HL,0000
6D08 22 00 6D      LD   (6D00),HL
6D0B ED 5B 00 6D   LD   DE,(6D00)
6D0F 4B            LD   C,E
6D10 5A            LD   E,D
6D11 1600          LD   D,00
6D13 210058        LD   HL,5800
6D16 0620          LD   B,20
6D18 19            ADD  HL,DE
6D19 10FD          DJNZ 6D18
6D1B 59            LD   E,C
6D1C 19            ADD  HL,DE
6D1D 22026D        LD   (6D02),HL
6D20 CBFE          SET  7,(HL)
6D22 3E00          LD   A,00
6D24 21085C        LD   HL,5C08
6D27 3600          LD   (HL),00
6D29 BE            CP   (HL)
6D2A 28FD          JR   Z,6D29
6D2C 7E            LD   A,(HL)
6D2D 32046D        LD   (6D04),A
6D30 2A026D        LD   HL,(6D02)
6D33 CBBE          RES  7,(HL)
6D35 11AC6D        LD   DE,6DAC
6D38 21046D        LD   HL,6D04
6D3B 1A            LD   A,(DE)
6D3C BE            CP   (HL)
```

```
6D3D 2007     JR   NZ,6D46
6D3F 13       INC  DE
6D40 EB       EX   DE,HL
6D41 5E       LD   E,(HL)
6D42 23       INC  HL
6D43 56       LD   D,(HL)
6D44 EB       EX   DE,HL
6D45 E9       JP   HL
6D46 13       INC  DE
6D47 13       INC  DE
6D48 13       INC  DE
6D49 FEFF     CP   FF
6D4B 20EE     JR   NZ,6D3B
6D4D ED5B006D LD   DE,(6D00)
6D51 3E16     LD   A,16
6D53 D7       RST  10
6D54 7A       LD   A,D
6D55 D7       RST  10
6D56 7B       LD   A,E
6D57 D7       RST  10
6D58 7E       LD   A,(HL)
6D59 D7       RST  10
6D5A 21006D   LD   HL,6D00
6D5D 34       INC  (HL)
6D5E 3E20     LD   A,20
6D60 BE       CP   (HL)
6D61 20A8     JR   NZ,6D0B
6D63 3600     LD   (HL),00
6D65 00       NOP
6D66 00       NOP
6D67 00       NOP
6D68 00       NOP
6D69 21016D   LD   HL,6D01
6D6C 3E15     LD   A,15
6D6E BE       CP   (HL)
6D6F CA0B6D   JP   Z,6D0B
6D72 34       INC  (HL)
6D73 C30B6D   JP   6D0B
6D76 21016D   LD   HL,6D01
6D79 3E00     LD   A,00
6D7B BE       CP   (HL)
6D7C CA0B6D   JP   Z,6D0B
6D7F 35       DEC  (HL)
6D80 C30B6D   JP   6D0B
6D83 21006D   LD   HL,6D00
6D86 35       DEC  (HL)
6D87 3EFF     LD   A,FF
6D89 BE       CP   (HL)
6D8A C20B6D   JP   NZ,6D0B
6D8D 361F     LD   (HL),1F
6D8F 00       NOP
6D90 00       NOP
6D91 C3766D   JP   6D76
6D94 ED5B006D LD   DE,(6D00)
6D98 3E16     LD   A,16
6D9A D7       RST  10
6D9B 7A       LD   A,D
6D9C D7       RST  10
```

```
6D9D 7B       LD   A,E
6D9E D7       RST  10
6D9F 3E20     LD   A,20
6DA1 D7       RST  10
6DA2 C3836D   JP   6D83
6DA5 C9       RET
6DA6 21006D   LD   HL,6D00
6DA9 C3636D   JP   6D63
```

Mettete i codici esadecimali in istruzioni DATA nel programma Hex loader in BASIC, e poi, subito dopo questi dati, inserite i seguenti codici esadecimali (però non anche gli indirizzi che sono riportati sulla sinistra!):

```
6DAC 07 05 6D 08 83 6D 09 5A
6DB4 6D 0A 69 6D 08 76 6D 0C
6DBC 94 6D 0D A6 6D 0F A5 6D
6DC4 FF 00 00 00 00 00 00 00
6DCC 00 00 00 00 00 00 00 00
6DD4 00 00 00 00 00 00 00 00
6DDC 00 00 00 00 00 00 00 00
6DE4 00 00 00 00 00 00 00 00
```

Adesso, poiché abbiamo iniziato usando gli indirizzi da 6D00 in poi, invece di 70 00 come facciamo di solito, dovremo cambiare la linea 10 con:

10 LET Indirizzo = 27909

Nel caso in cui abbiate fatto un errore con le istruzioni DATA, è meglio conservare su nastro il loader e i dati esadecimali prima di far girare il programma. Quando l'avete fatto potrete far girare il loader con RUN e poi iniziare a scrivere con Typewriter con il comando:

PRINT AT 0,0;: RANDOMIZE USR 27909.

Dato che abbiamo esaminato in dettaglio il programma, dovreste già sapere come usarlo, ma se non è così ecco come fare:

SHIFT 5 - SHIFT 8	—	Controlli di cursore
DELETE	—	Cancella il carattere
EDIT	—	Riporta il cursore alla posizione 0,0
GRAPHICS	—	Ritorno al BASIC.

Buon divertimento!

CAPITOLO 4

LE OPERAZIONI LOGICHE

Abbiamo visto, nel capitolo dedicato ai bit, le principali istruzioni per manipolarli in modo individuale. In questo capitolo vedremo ancora alcune istruzioni sui bit che ci permettono di modificarli uno alla volta. Le istruzioni che vedremo sono chiamate "booleane", dato che si servono della logica "booleana". Può essere che voi non abbiate ancora alcun'idea di cosa sia la logica "booleana": si tratta semplicemente di un termine usato per indicare l'insieme delle istruzioni logiche, proprio come il termine "linguaggio strutturato" serve a descrivere un linguaggio con struttura come quella del Pascal o del FORTH.

In BASIC ci sono le parole OR, AND e NOT. Le usiamo nelle istruzioni IF... THEN... per prendere una decisone. Anch'esse fanno parte della logica booleana, ma in modo molto più flessibile. Non è possibile produrre un equivalente in linguaggio macchina all'istruzione IF... THEN..., ma si può ottenere un effetto molto simile usando ciò che già sappiamo riguardo ai salti condizionati (come per esempio JP Z, nnnn) insieme con queste "istruzioni logiche". Vediamo brevemente il principio che sta alla base delle istruzioni OR e AND del BASIC.

Come già sappiamo, queste istruzioni funzionano così:

1. X AND Y : se la condizione x è vera e la condizione y è vera, il risultato è vero. Per esempio 1=1 AND 10=10 è vero, e "H"="H" AND a$=a$ è pure vero, mentre 2=3 AND 1=1 è falso e "H"="H" AND "B"="C" A è falso.

2. x OR y: se è vera la condizione x o la condizione y o ambedue, il risultato è vero. Ad esempio, 2 = 2 OR 3 è vero, a$ = a$ OR b$ = b$ è vero, mentre 1 = 2 OR 5 = 0 è falso, e "C" = "D" OR "B" = "A" è falso.

In linguaggio macchina non si lavora con stringhe di caratteri o con variabili, bensì con un bit, ovvero "zeri" e "uni". Possiamo farlo fare anche al BASIC e il computer ci dirà se qualcosa è vero o falso. Provate a scrivere:

PRINT 1=1

Ciò può sembrare un po' stupido, tuttavia il computer risponde con un "1". Perché? Perché scrivendo "1" esso ci dice che la condizione 1=1 è vera. Provate ora:

PRINT 1=0

Sappiamo che ciò non è vero, e lo sa anche lo Spectrum che ce lo fa sapere scrivendo "0". Possiamo così concludere che quando si usano gli "uni" e gli "zeri":

0 corrisponde a falso
1 corrisponde a vero.

Provate ora a scrivere:

LET false = 0: LET true = 1

PRINT true AND true

Ma poiché true=1 il computer ha scritto "1". Questo perché la variabile "true" è equivalente ad un'espressione vera. Ricordando che false = 0 e che 0 in effetti sta per falso, potete prevedere cosa succederebbe se scriveste:

PRINT false AND true.

Viene stampato uno 0 (con significato di falso), perché le due condizioni non sono entrambe vere. Possiamo preparare una tabellina per l'istruzione AND:

AND		RISULTATO
falso	falso	falso
vero	vero	vero
falso	vero	falso
vero	falso	falso

Possiamo ora convertirla ad una tabella con "uni" e "zeri", ricordando che 1 significa vero e Ø falso:

AND:			
	Ø	Ø	Ø
	Ø	1	Ø
	1	Ø	Ø
	1	1	1

Facendo gli stessi esperimenti con l'istruzione OR troveremo:

OR:			
	Ø	Ø	Ø
	Ø	1	1
	1	Ø	1
	1	1	1

Queste tabelle corrispondono esattamente alle operazioni eseguite dalle istruzioni OR e AND del linguaggio macchina, tranne il fatto che esse operano non soltanto su una coppia di bit, ma su un insieme di otto coppie di bit, ovvero due byte.

Se usiamo l'istruzione AND sull'accumulatore, la CPU controllerà tutti i bit uno alla volta, confrontando i bit corrispondenti nei due registri. Il risultato di ogni confronto sarà 1 o Ø decidendo in base alla tabella che abbiamo appena visto. Per esempio:

```
AND:        1 1 Ø 1 Ø 1 1 Ø
            Ø 1 1 Ø 1 1 Ø 1
            _______________
risultato   Ø 1 Ø Ø Ø 1 Ø Ø
```

Notate come gli unici bit che nel formato sono posti a uno siano quelli con sopra due bit uguali a uno. Guardate quest'altro esempio e seguitelo bit per bit, cercando di capirne il risultato.

```
AND:        Ø Ø 1 1 1 1 Ø 1
            1 1 1 1 Ø 1 Ø Ø
            _______________
risultato   Ø Ø 1 1 Ø 1 Ø Ø
```

Avete capito? allora provate a farlo da soli:

AND: 1 Ø 1 1 Ø 1 1 1
risultato 1 Ø 1 Ø 1 1 Ø 1

?

Ricordando che soltanto 1 e 1 danno 1 come risultato, dovreste ottenere 1Ø1ØØ 1Ø1. Ecco di nuovo la tabella OR:

OR:

Ø	Ø	Ø
1	Ø	1
Ø	1	1
1	1	1

Quindi, operando con OR sui due byte seguenti:

Ø 1 Ø 1 1 Ø Ø 1
Ø 1 1 1 Ø 1 Ø 1

otteniamo:

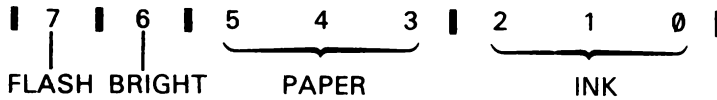
Ø 1 1 1 1 1 Ø 1

Notate che gli unici bit che nel risultato hanno Ø sono quelli che sopra hanno due bit pari a zero.

L'istruzione AND in linguaggio macchina prende la forma di "AND r", dove r è un registro. Essa compie l'operazione logica AND sul registro r con l'accumulatore, e mette il risultato nell'accumulatore stesso. Analogamente "OR r" compie l'operazione logica OR nel registro r con l'accumulatore.
Ecco la lista completa delle istruzioni AND e OR con i relativi codici esadecimali:

AND A	A7
AND B	AØ
AND C	A1
AND D	A2
AND E	A3
AND H	A4
AND L	A5
AND (HL)	A6

AND (IX + dis)	DD A6 dis
AND (IY + dis)	FD A6 dis
AND nn	E6 nn
OR A	B7
OR B	BØ
OR C	B1
OR D	B2
OR E	B3
OR H	B4
OR L	B5
OR (HL)	B6
OR (IX + dis)	DD B6 dis
OR (IY + dis)	ED B6 dis
OR nn	F6 nn

L'utilizzo di AND è quello di controllare il valore di certi bit di un byte. Prendiamo un esempio da un byte di attributi, come abbiamo fatto nel terzo capitolo.

Il nostro compito è quello di cambiare tutte le posizioni di caratteri da INK Ø a INK 2 (cioè da nero a rosso). Ovviamente avremo bisogno di un ciclo per elaborare ciascun byte di attributi. Ma l'istruzione cruciale sarà all'interno del ciclo. Anzitutto vogliamo prendere i tre bit meno significativi, ossia i bit Ø, 1 e 2, per controllarne il valore. Per fare ciò useremo l'istruzione AND. Considerate questo esempio:

```
ATTRIBUTE 1 1 Ø 1 Ø 1 1 1
Ø7        Ø Ø Ø Ø Ø 1 1 1
          _______________     operazione AND
RESULT    Ø Ø Ø Ø Ø 1 1 1
```

Noterete che soltanto i bit Ø, 1 e 2 compaiono nel risultato. Ciò è dovuto al fatto che si è usata l'istruzione AND sul byte di attributi insieme ad un byte contenente il numero Ø7 Hex, ovvero con i bit Ø, 1 e 2 pari a uno. Guardate quest'altro esempio:

```
ATTRIBUTE 1 1 1 1 Ø 1 1 Ø
Ø7        Ø Ø Ø Ø Ø 1 1 1
          _______________     operazione AND
RESULT    Ø Ø Ø Ø Ø 1 1 Ø
```

Il risultato è 06 Hex, ovvero il numero rappresentato dal valore dei bit 0, 1 e 2. Possiamo dire dunque che operando con AND su un numero e un byte, i bit pari a zero "mascherano" i bit corrispondenti nell'altro byte. Non ci sorprendiamo allora se questa procedura viene chiamata "masking" (mascheramento), poiché descrive con piena efficacia il modo in cui riusciamo a nascondere alcuni bit di un byte.

Tornando al nostro compito, se operiamo con AND 07 Hex sul byte di attributi, rimaniamo con il solo colore INK, dato che gli altri bit vengono tutti posti a zero dall'istruzione AND. Potremo quindi fare un test sul numero di INK usando i segnali e vedere se lo dobbiamo cambiare:

```
7000 210058   LD    HL,5800
7003 7E       LD    A,(HL)
7004 E607     AND   07
7006 2002     JR    NZ,700A
7008 CBCF     SET   1,A
700A 77       LD    (HL),A
700B 23       INC   HL
700C 7C       LD    A,H
700D FE5B     CP    5B
700F 20F2     JR    NZ,7003
7011 C9       RET
```

Considerando il programma passo dopo passo, vediamo anzitutto che in HL viene caricato l'indirizzo d'inizio del file degli attributi. Poi nell'accumulatore viene caricato l'attributo cui punta HL. Poi operiamo con AND 07, ed azzeriamo tutti i bit tranne quelli relativi al colore INK. Se il byte risultante è diverso da zero, si esegue un salto relativo alla locazione 700A, dato che se il byte non è zero il colore INK non può essere nero. Se invece INK è nero, poniamo ad uno il bit 1 dell'accumulatore, rendendo rosso il colore INK.

Controlliamo che il programma funzioni, assicuriamoci che il loader in BASIC sia pronto; poi cambiate la linea 80 con i codici riportati nel listato del programma.

Infine scrivete RUN; seguito da
INK 2: LIST: RANDOMIZE USR indirizzo

Se il colore PAPER non è rosso, vedrete che il listato in nero immediatamente diventa rosso! Per verificare che vengono modificati soltanto i caratteri in nero, scrivete

INK 0: LIST: INK2: LIST: RANDOMIZE USR indirizzo

Un utilizzo frequente dell'istruzione OR è quello di porre a uno certi particolari bit di un byte. Per esempio abbiamo un byte di attributi e vogliamo porre a uno i bit sei e sette per rendere il carattere brillante e lampeggiante. Possiamo farlo

creando un byte che abbia pari a uno soltanto i bit sei e sette, e poi, applicando l'istruzione OR a questo byte insieme al byte di attributi. Mettendo questa procedura all'interno di un ciclo possiamo far lampeggiare con caratteri brillanti l'intero schermo.

```
ATTRIBUTE 0 0 1 0 1 0 0 1
          1 1 0 0 0 0 0 0
          _______________   operazione OR
          1 1 1 0 1 0 0 1
```

Notate come gli unici bit che vengono cambiati sono i bit sei e sette, cioè proprio quello che volevamo. Vediamo allora il programma che fa lampeggiare e brillare l'intero schermo.

```
7000 210058   LD    HL,5800
7003 7E       LD    A,(HL)
7004 F6C0     OR    C0
7006 77       LD    (HL),A
7007 23       INC   HL
7008 7C       LD    A,H
7009 FE5B     CP    5B
700B 20F6     JR    NZ,7003
700D C9       RET
```

La procedura è molto semplice: HL viene puntato all'inizio del file di attributi, poi un attributo viene caricato in A; i bit sei e sette vengono posti a uno, mediante l'istruzione OR CO, e l'attributo viene memorizzato di nuovo. Infine, HL viene aumentato di uno, si fa un test per vedere se ha raggiunto la fine ed in caso negativo si procede con l'attributo successivo. Verifichiamo ora che il programma funzioni. Usate la seguente linea 80 nel loader:

80 DATA "2100587EF6C077237CFE5B20F6C9"

Poi, dopo aver scritto RUN scrivete

LIST: RANDOMIZE USR indirizzo

e potrete vedere il cambiamento istantaneo dello schermo che diventa brillante e lampeggiante.

COSE DA PROVARE

1. Provate valori diversi da CO nell'istruizone OR e cercate di chiarirvi i motivi degli effetti ottenuti.
2. Provate ad usare AND invece di OR, con diversi valori. Chiaritevi i motivi degli effetti risultanti.

3. Invece di usare OR per porre a uno i bit sei e sette, modificate il programma usando l'istruzione SET che abbiamo visto nel precedente capitolo.

ESCLUSIVO!

Esiste un'altra istruzione logica da considerare, essa è chiamata OR Esclusivo ed è molto simile alla normale istruzione OR. Viene usata dallo Specturm quando si seleziona l'opzione OVER. Considerate la seguente tabella:

PAPER + PAPER = PAPER
INK + PAPER = INK
PAPER + INK = INK
INK + INK = PAPER

Questi sono i risultati quando si seleziona OVER. L'istruzione XOR viene quando si disegna o si scrive sul video. Se immaginiamo che INK sia "1" e PAPER sia "0 notiamo che:

XOR	
00	0
10	1
01	1
11	0

Questa è la tabella logica per l'istruzione XOR; l'unica differenza rispetto a quella dell'istruzione OR è alla fine, dove due "uno" danno come risultato zero invece di uno. Ecco ora la lista dei codici esadecimali di tutte le istruzioni XOR:

XOR A	AF
XOR B	A8
XOR C	A9
XOR D	AA
XOR E	AB
XOR H	AC
XOR L	AD
XOR (HL)	AE
XOR (IX + dis)	DD AE dis
XOR (IY + dis)	FD AE dis
XOR nn	EE nn

Provate ad usare XOR nell'ultimo programma visto e vedete quali sono gli effetti. Cercate anche di provarlo quando c'è qualche carattere che sta già lampeggiando

o è brillante sul video. Se non capite il perché di questi effetti, calcolatevi a mano queste “addizioni” e forse così li capirete. Se avete qualche difficoltà servitevi della seguente tabella:

XOR	Ø	Ø	Ø
	Ø	1	1
	1	Ø	1
	1	1	1

1. 1 XOR Ø = ?

2. 1 XOR 1 = ?

```
3.    1 Ø 1 1          4.    1 1 Ø Ø
XOR   Ø 1 1 Ø                1 Ø 1 Ø
      -------                -------
         ?                      ?

5.    1 Ø 1 1 1 Ø 1 1
XOR   Ø 1 1 Ø 1 Ø 1 Ø
      ---------------
         ?
```

L’ultimo argomento che tratteremo in questo capitolo è un metodo per il “test di azzeramento di una coppia di byte”. Nel secondo capitolo abbiamo visto come preparare un ciclo che utilizza registri ad un solo byte. Quando però vogliamo eseguire una operazione più di 256 (decimale) volte dobbiamo usare come contatore un registro a due byte. Un modo per fare il ciclo è usando l’istruzione CP.

```
        LD HL,1ØØØ
PROG         :
             :
        LD A,H
        CP ØØ
        JR NZ,PROG
        LD A,L
        CP ØØ
        JR NZ,PROG
        RET
```

Come noterete H ed L vengono sottoposti a test separatamente per vedere se sono entrambi nulli. L'istruzione CP non richiede due byte: possiamo risparmiare spazio usando invece l'istruzione AND. Se facciamo "AND A" l'accumulatore non viene modificato mentre la cosa più importante è che i vari esgnali sono influenzati dall'operazione. Vediamo perché A non viene modificato: ricordate infatti che l'istruzione "AND A" non fa altro che compiere l'operazione AND sull'accumulatore con se stesso, e quindi i due numeri sono uguali.

```
1 1 0 0 1 0 1 0
1 1 0 0 1 0 1 0
---------------
1 1 0 0 1 0 1 0
```

Tenete sempre presente la tabella per l'operazione AND.

```
0 0 0
-----
0 1 0
-----
1 0 0
-----
1 1 1
```

Quindi, come possiamo ben vedere, A viene lasciato inalterato, ma i segnali risultano influenzati dall'operazione. Ciò ci permette di prendere una valida decisione, e possiamo perciò sostituire le istruzioni CP00 con AND A:

```
        LD HL,1000
PROG        :
        INC HL
        LD A,H
        AND A
        JR NZ,PROG
        LD A,L
        AND A
        JR NZ,PROG
        RET
```

Ma così non abbiamo ancora ottenuto un risultato molto significativo. Possiamo ridurre ulteriormente questo test di azzeramento usando l'istruzione OR. Se nel registro A carichiamo H e lo confrontiamo con L mediante l'istruzione OR, qualunque bit diverso da zero comparirà immediatamente, dato che A risulterà

diverso da zero. Se invece tutti i bit di HL sono zero, anche il risultato in A sarà nullo. Ecco allora come si presenta il ciclo modificato:

```
PROG     LD HL,1000      21 00 10
START         :
         INC:HL          23
         LD A,H          7C
         OR L            B5
         JR NZ START     20?
         RET             C9
```

Facciamo una prova, usando questa procedura per stampare 200 (Hex) asterischi sul video.

```
7000 210020   LD   HL,2000
7003 3E2A     LD   A,2A
7005 D7       RST  10
7006 23       INC  HL
7007 7C       LD   A,H
7008 B5       OR   L
7009 20FA     JR   NZ,7005
700B C9       RET
```

Mettete i codici esadecimali riportati sulla destra nelle istruzioni DATA, con le virgolette, nel nostro programma Hex-loader. Poi scrivete RUN, e infine

PRINT AT 0,0; RANDOMIZE USR indirizzo

Vedrete che esattamente 200 (Hex), ovvero 512 decimale, asterischi verranno stampati sullo schermo.

CAPITOLO 5

ROTAZIONI

Questo capitolo è tutto dedicato alle rotazioni. Ciò che viene effettuato da queste istruzioni in fondo non è altro che spostare verso destra o verso sinistra i bit di un registro, con varie combinazioni per quanto riguarda il segnale di riporto. Esse sono meglio rappresentate in modo grafico, ma ho dato anche una spiegazione scritta per ognuna di esse, insieme ai codici esadecimali corrispondenti. Le istruzioni di rotazione non hanno uno scopo molto evidente, ma possono essere usate nelle applicazioni in cui è necessario costruire un byte bit dopo bit, oppure quando si vuole effettuare un test su un byte considerando un bit alla volta. Le istruzioni di rotazione possono anche essere usate per le moltiplicazioni di potenze di 2 (2, 4, 6, 8 etc.), e ci sono anche un paio di istruzioni, che possono essere usate con il BCD, "Binary Coded Decimal", e che spiegherò più avanti.

La maggior parte di queste istruzioni hanno una versione che funziona in modo specifico con l'accumulatore, e una serie di altre versioni, ma per ciascuna dei soliti registri ad un solo byte, incluso anche l'accumulatore. Queste ultime richiedono due byte di codice operativo, mentre la prima versione solo per l'accumulatore richiede un solo byte. Per l'accumulatore esistono dunque due istruzioni equivalenti, con l'unica differenza che una delle due è di un solo byte, mentre l'altra è lunga 2 byte. Ovviamente, allo scopo di risparmiare memoria, quando si tratta di dover ruotare i bit dell'accumulatore si preferirà usare la versione ad un solo byte.

RLC r (r=A,B,C,D,E,H,L,(HL),(IX + dis),(IY + dis))

Questa istruzione effettua una rotazione del registro o del byte "r" verso sinistra: il bit più significativo, ovvero il bit sette, viene portato al posto del bit meno significativo, e tutti gli altri bit vengono spostati di una posizione verso sinistra. Il bit, o segnale, di riporto viene posto a uno o a zero per registrare il contenuto del bit sette prima della rotazione. Ciò permette al programmatore di conoscere lo stato del bit sette prima dell'operazione di spostamento, effettuando un test sul segnale di riporto dopo l'istruzione di rotazione.

RLCA	Ø7	RLC H	CB Ø4
		RLC L	CB Ø5
RLC A	CB Ø7	RLC (HL)	CB Ø6
RLC B	CB ØØ	RLC (IX + dis)	DD CB dis Ø6
RLC C	CB Ø1	RLC (IY + dis)	FD CB dis Ø6
RLC D	CB Ø2		
RLC E	CB Ø3		

RL r

Questa istruzione fa ruotare verso sinistra il registro o byte indicato con "sette", servendosi anche del bit di riporto: il bit 7 viene spostato nel bit di riporto e il precedente contenuto del bit di riporto viene spostato nel bit zero. Tutti gli altri bit vengono spostati verso sinistra.

RLA	17	RL E	CB 13
		RL H	CB 14
RL A	CB 17	RL L	CB 15
RL B	CB 1Ø	RL (HL)	CB 16
RL C	CB 11	RL (IX + dis)	DD CB dis 16
RL D	CB 12	RL (IY + dis)	FD CB dis 16

RRC r

Questa operazione è analoga a quella compiuta dall'istruzione RLC r tranne per il fatto che la rotazione viene compiuta verso destra. Questa volta è il bit zero che viene spostato al bit di "r" ed anche ricopiato nel bit di riporto.

RRCA	ØF	RRC E	CB ØB
		RRC H	CB ØC
RRC A	CB ØF	RRC L	CB ØD
RRC B	CB Ø8	RRC (HL)	CB ØE
RRC C	CB Ø9	RRC (IX + dis)	DD CB dis ØE
RRC D	CB ØA	RRC (IY + dis)	FD CB dis ØE

RR r

Questa infine è l'operazione inversa di RL r: qui il bit zero di "r" viene portato nel bit di riporto, il precedente valore del bit di riporto viene messo nel bit sette di "r", e tutti gli altri bit vengono spostati verso destra di una posizione.

RRA	1F	RRE	CB 1B
		RR H	CB 1C
RR A	CB 1F	RR L	CB 1D
RR B	CB 18	RR (HL)	CB 1E
RR C	CB 19	RR (IX + dis)	DD CB dis 1E
RR D	CB 1A	RR (IY + dis)	FD CB dis 1E

A volte è preferibile effettaure un'operazione di spostamento anziché una di rotazione in un registro: invece di far ruotare il bit zero e portarlo al bit sette o viceversa, il bit sette può essere azzerato o lasciato inalterato mentre il suo valore originario viene spostato al bit successivo. Queste istruzioni di spostamento dei bit non hanno una seconda versione ad un solo byte per l'accumulatore: esse sono tutte formate da due byte di codice esadecimale di cui il primo è sempre CB (Hex).

SLA r

Questa istruzione sposta tutti i bit del registro o del byte "r" verso sinistra. Il bit 7 viene portato nel bit di riporto ed il bit zero viene azzerato.

SLA A	CB 27	SLA H	CB 24
SLA B	CB 20	SLA L	CB 25
SLA C	CB 21	SLA (HL)	CB 26
SLA D	CB 22	SLA (IX + dis)	DD CB dis 26
SLA E	CB 23	SLA (IY + dis)	FD CB dis 26

SRA r

Questa istruzione sposta tutti i bit del registro o del byte "r" verso destra. Il bit zero viene messo nel bit di riporto. Il bit 7 viene copiato nel bit 6 ma non viene modificato.

SRA A	CB 2F	SRA H	CB 2C
SRA B	CB 28	SRA L	CB 2D
SRA C	CB 29	SRA (HL)	CB 2E
SRA D	CB 2A	SRA (IX + dis)	DD CB dis 2E
SRA E	CB 2B	SRA (IY + dis)	FD CB dis 2E

SRL r

Questa istruzione serve per un'operazione di spostamento verso destra ma anche se la cosa può sembrare insolita non esiste un suo corrispettivo di spostamento verso sinistra, ad esempio SLL r. C'è persino un intervallo di codici inutilizzati dove potrebbero stare i codici per le istruzioni SLL, il che indica come effettivamente questa sia una funzione del microprocessore Z80 che la ditta Zilog, che ha

prodotto il chip, non è riuscita a rendere operativa. L'istruzione SRL serve a spostare i bit verso destra, azzerando il bit 7 e muovendo il bit Ø nel bit di riporto.

SRL A	CB 3F	SRL H	CB 3C
SRL B	CB 38	SRL L	CB 3D
SRL C	CB 39	SRL (HL)	CB 3E
SRL D	CB 3A	SRL (IX + dis)	DD CB dis 3E
SRL E	CB 3B	SRL (IY + dis)	FD CB dis 3E

NUMERI DECIMALI

Prima di prendere in considerazione le istruzioni RLD e RRD, dobbiamo capire come funziona il sistema BCD "Binary Coded Decimal", ovvero "decimali con codifica binaria". Il sistema BCD è un argomento molto importante nella teoria dei computer. Si tratta di una specie di interfaccia tra uomini e il sistema binario, che cerca di collegare i due sistemi di numerazione decimale e binario. La maggior parte degli indicatori a LED a sette segmenti ricevono le cifre sotto forma di cifra BCD. Una singola cifra BCD è molto semplice da capire: si tratta di quattro bit il cui valore d'assieme non supera 9, altrimenti non saremmo più nel sistema decimale.

Quando vogliamo memorizzare un numero in forma decimale possiamo usare il sistema BCD. Per esempio, se volessimo memorizzare 99 come numero decimale, lo dovremmo prima convertire in esadecimale e poi riconvertirlo di nuovo se lo volessimo stampare sul video come decimale.
Ecco come verrebbe rappresentato il numero 56 decimale:

quattro bit più significativi Ø1Ø1 Ø11Ø quattro bit meno significativi
5 6

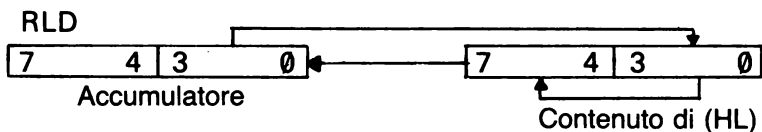
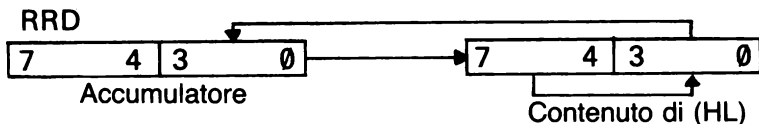
Se dovessimo prendere invece questo 56 come un numero esadecimale e poi convertirlo, troveremmo (guardando il numero 56 Hex nell'appendice del manuale) che corrisponde all'86 decimale, mentre noi stavamo considerando il 56 come un numero decimale, non esadecimale.

Se un normale numero esadecimale è stato caricato nell'accumulatore, lo si può convertire al numero decimale BCD mediante l'istruzione DAA, "Decimal Adjust Accumlator", ovvero accumulatore ad aggiustamento decimale. Il codice esadecimale per l'istruzione DAA è 27 Hex.

Immaginiamo allora di aver caricato nell'accumulatore il numero 43 Hex.
Guardando nel manuale troviamo che il 43 esadecimale corrisponde al 67

decimale. Se ora facciamo eseguire l'istruzione DAA, il valore dell'accumulatore passa da 43 a 67, memorizzato come numero BCD.

L'istruzione DAA può modificare in questo modo numeri al massimo fino a 99 decimale, dato che in un registro ci sono solo due cifre decimali. Se sommiamo due numeri BCD il risultato dovrà essere aggiustato e l'istruzione DAA può servire per correggere il risultato portandolo alla codifica BCD.

Vediamo ora come funzionano le istruzioni RLD e RRD.

L'istruzione RLD consente di ruotare verso sinistra i seguenti tre blocchi di quattro bit ciascuno: la metà inferiore dell'accumulatore e le due metà della locazione di memoria su cui è puntato il registro HL. Possiamo immaginare che questi blocchi da quattro bit siano cifre decimali nel sistema BCD. Un altro nome, meno noto, per indicare mezzo byte è il termine "nibble": i patiti dell'informatica potranno confermarlo (già che ci troviamo in argomento di strani termini tecnici, lo sapevate che nei manuali originali del PET della Commodore si fa riferimento ad 1/50 di secondo come a un "jiffy"?). Torniamo all'argomento in discussione. Eccovi ora uno schema per l'istruzione RRD.

L'istruzione RRD funziona nella direzione opposta a quella di RLD, spostando la cifra 13 CD più significativa di (HL) nella metà meno significativa, e la metà meno siginficativa nella metà inferiore dell'accumulatore e la metà inferiore dell'accumulatore nella metà più significativa di (HL).

Possiamo così concludere il quinto capitolo. Nel prossimo capitolo le cose suoneranno molto bene, le navi andranno avanti e indietro, ed i porti saranno ravvivati dal suono di musica. Siete confusi? Continuate a leggere...

CAPITOLO6

LE PORTE

I computer devono essere in grado di comunicare con il mondo esterno, ovvero con l'utente, con le stampanti, i registratori a nastro, le tastiere e i video che fanno parte del mondo dei computer ZX. Senza la capacità di comunicare un computer sarebbe inutile, un po' come acquistare un video registratore senza avere anche il televisore. Possiamo distinguere le comunicazioni di un computer in due classi: quelle in ingresso e quelle in uscita. L'ingresso è dato dalle informazioni ricevute dal computer, come ad esempio quelle dalla tastiera su cui danzano allegramente le vostre dita! Viceversa l'uscita è data dalle informazioni emesse dal computer, come sullo schermo del televisore. Per consentirgli di comunicare con i vari dispositivi di ingresso e di uscita, un normale sistema computerizzato, all'interno di una lavatrice programmabile così come nello Spectrum ZX, viene dotato delle cosiddette porte, o "porti". Questo termine tecnico possiede una certa sua logica: si può infatti immaginare che ci sia una nave che trasporta le informazioni nel porto affinché il computer le possa ricevere e che poi carichi altre informazioni dal computer per portarle altrove. Ovviamente guardando dentro il vostro Spectrum non vedrete delle navi che si muovono, ma il principio è molto simile. Se siete dei maniaci dell'hardware, potrete usare alcune delle porte libere dello Spectrum; prima però conviene che vi leggiate il capitolo 23 del manuale Sinclair così evito di ripetere le ottime spiegazioni di Steven Vickers.

Tuttavia le porte non vengono utilizzate soltanto dai maniaci dell'elettronica. Non c'è bisogno di comprare un'attrezzatura aggiuntiva per vedere come funzionano, dato che abbiamo a portata di mano alcune porte molto utili, quelle che lo Spectrum utilizza per comunicare con il mondo esterno. Spero che abbiate già un'idea di come avere accesso alle porte del BASIC, ma comunque inizieremo proprio da lì dato che costituisce una buona introduzione per capire l'aspetto della questione legato al linguaggio macchina.
Facciamo dunque un breve riepilogo sulle porte. Normalmente ci possono essere fino a 256 porte diverse su un sistema Z8Ø, ma sfruttando una peculiarità dello Z8Ø della Zilog, si può a volte arrivare fino a 65536 porte, come vedremo più avanti nel capitolo.

Lo Spectrum si serve soltanto di alcune di queste porte: quella che noi useremo è la porta che consente di predisporre il colore della cornice e per produrre suoni. Provate a scrivere questo breve programma e fatelo eseguire:

```
  10 FOR i=0 TO 7: OUT 254,i: PA
USE 3: NEXT i: GO TO 10
```

Come potete vedere, essa fa cambiare il colore della cornice passando attraverso tutti gli otto colori diversi dello Spectrum. La prima cosa da notare è che stiamo utilizzando la porta numero 254: questa porta è associata alla cornice e con la presa per l'altoparlante, cuffia e microfono. L'istruzione OUT del BASIC ha la seguente forma generale:

OUT porta, dati.

Nel nostro esempio la porta è il numero 254 e i dati sono i colori per la cornice. La stessa porta è suddivisa in varie parti, ciascuna delle quali viene controllata da uno o più bit degli otto bit che compongono un byte di dati. Ecco come operano i bit nella porta 254:

0	)	
1	)	Colore del bordo
2	)	
3	—	presa microfono
4	—	altoparlante
5	)	
6	)	inutilizzati
7	)	

Ora che sappiamo come funziona la cosa, dovremmo essere in grado di generare qualche suono. Anzitutto dobbiamo renderci conto che un 1 nel bit 4 fa sì che la membrana dell'altoparlante venga spinta in fuori, mentre uno 0 fa sì che essa venga spinta in dentro. Dunque se alterniamo gli uni con gli zeri dovremmo ricavarne un certo rumore. Per fare ciò, scrivete il seguente programma:

```
  10 OUT 254,2↑4*1: OUT 254,2↑4*
0: GO TO 10
```

Ci sono due comandi OUT, uno per inviare un 1 all'altoparlante, l'altro per inviargli uno 0. Guardando il listato noterete che il primo dato è "2^4*1": il 2 c'è perché stiamo lavorando in base binaria cioè base due; il 4 sta ad indicare che il dato

si riferisce al quarto bit, e la cifra 1 serve perché vogliamo che la membrana dell'altoparlante sia spinta in fuori. Analogamente nel secondo comando OUT, dove però c'è uno 0 perché vogliamo che la membrana venga spinta in dentro. Così com'è, il programma è molto lento ed il suono prodotto non è altro che fievoli "scatti". Per aumentare la velocità dobbiamo evitare che il computer si calcoli ogni volta le due espressioni: possiamo quindi sostituire "2^4*1" e "2^4*0 " con i loro effettivi valori, 16 e 0 rispettivamente. Modificando il programma avremmo:

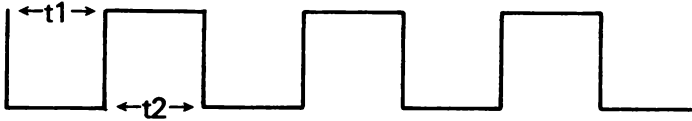
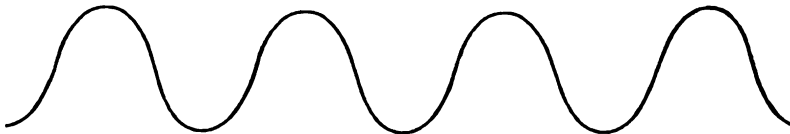
```
  10 OUT 254,16: OUT 254,0: GO T
O 10
```

Ma anche con questa modifica il ronzio risulta molto tenue e molto lontano dagli ZAP, POW e BOOM di una macchina per il gioco "Pacman" o "Defenders". Allora, per aumentare un po' il ritmo trasferiamo i nostri sforzi al linguaggio macchina, visto che già abbiamo una certa conoscenza di base circa le porte.
Nel linguaggio macchina dello Z80, ci sono istruzioni che corrispondono direttamente a quelle del BASIC. Per esempio, per inviare alla porta 254 i dati contenuti nell'accumulatore, basterà dire:

OUT (FE), A.

Diversamente dal BASIC qui dovremo mettere fra parentesi il numero della porta: FE, ovvero l'equivalente Hex di 254 decimale. Si fa così perché il numero della porta viene visto come un indirizzo e dunque, per seguire il formato standard dello Z80, usiamo le parentesi per indicare che l'informazione, in questo caso A, viene inviata ad una locazione, cioè l'indirizzo di una porta. Cerchiamo allora di tradurre in linguaggio macchina il nostro programma in BASIC.
Ora, poiché il linguaggio macchina è molto rapido dovremo inserire alcuni intervalli fra le varie istruzioni OUT, altrimenti non appena la membrana dell'altoparlante inzia a muoversi in una direzione riceve subito il segnale dal computer di tornare indietro, e l'effetto netto è che rimane a tremolare in una posizione intermedia, senza produrre alcun suono. Per questi intervalli di pausa il registro B è il più conveniente da usare. Fisseremo nel registro C il numero di scatti da fare. Il nome corretto per questi scatti è "cicli". Ecco il programma: per usarlo mettete i codici esadecimali nel programma loader in BASIC e seguite la solita procedura. Quelle due strane istruzioni all'inizio e alla fine non sono errori, ma si tratta di due istruzioni che verranno spiegate nel prossimo capitolo. In sostanza esse servono ad assicurare che il computer dedichi tutte le proprie energie alla generazione del suono e che non scappi altrove ogni 1/50 di secondo per controllare cose come la tastiera etc.

```
7000 F3       DI
7001 0EFF     LD    C,FF
7003 3E00     LD    A,00
7005 D3FE     OUT   (FE),A
7007 06C0     LD    B,C0
7009 10FE     DJNZ  7009
700B 3E18     LD    A,18
700D D3FE     OUT   (FE),A
700F 06C0     LD    B,C0
7011 10FE     DJNZ  7011
7013 0D       DEC   C
7014 20ED     JR    NZ,7003
7016 FB       EI
7017 C9       RET
```

Nel programma abbiamo posto in C il valore FF Hex, cosicché verranno prodotti 255 (dec) cicli. Ciò è in relazione con la durata del segnale. I due cicli che utilizzano il registro B servono invece a decidere per quanto tempo la membrana dell'altoparlante debba venire spinta in fuori e per quanto tempo debba venire spinta in dentro. Alterando questi valori di B possiamo aggiustare la frequenza del segnale. Ecco una raprpesentazione grafica del segnale:

La lunghezza di t1 viene fissata nel primo ciclo con B, e la lunghezza di t2 nel secondo ciclo con B. Il numero dei cicli viene fissato con C. Come potete vedere i cicli hanno forma quadrata, e quindi li chiameremo "onde quadre". Suoni diversi corrispondono a onde di forma diversa, un flauto per esempio solitamente produce una graziosa onda come questa:

Si tratta di un suono molto più arrotondato e dolce. Sfortunatamente con lo Spctrum siamo del tutto limitati all'ambito delle onde quadre, a meno che non aggiungiamo un dispositivo musicale; quindi il suono che riusciamo a produrre è sempre piuttosto duro e non molto melodioso. Ma anche con questa limitazione è possibile produrre effetti molto simpatici.

Un'ultimo appunto riguardo il nostro ultimo programma: avrete notato che la cornice diventa nera. Ciò dipende dall'aver lasciato a zero i bit 0, 1 e 2, il che fa risultare zero il colore della cornice, cioè nero!

Così come l'abbiamo scritto il nostro programma va bene; in realtà però esso non è necessario, poiché nella ROM c'è una routine che può fare quanto ci serve e che è molto semplice da usare. Essa è posta all'indirizzo 03B5 e per usarla dobbiamo fornirle il numero dei cicli in DE e la lunghezza dei cicli, ovvero la frequenza in HL. Per esempio, provate questo programma:

```
7000 210003   LD   HL,0300
7003 110002   LD   DE,0200
7006 CDB503   CALL 03B5
7009 C9       RET
```

Ecco qua! Un modo molto semplice per generare un semplice BIP! Ma noi vogliamo qualcosa di più di un semplice BIP, dato che già lo possiamo fare dal BASIC. Vediamo allora come possiamo produrre suoni più interessanti.

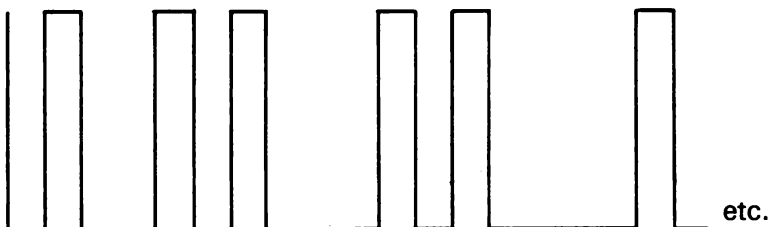
Il seguente programma usa la routine nella ROM per produrre un BIP con frequenza crescente:

```
7000 210010   LD   HL,1000
7003 112000   LD   DE,0020
7006 ED52     SBC  HL,DE
7008 110100   LD   DE,0001
700B E5       PUSH HL
700C CDB503   CALL 03B5
700F E1       POP  HL
7010 7C       LD   A,H
7011 A7       AND  A
7012 20EF     JR   NZ,7003
7014 C9       RET
```

Il programma inzia caricando 1000 (Hex) nel registro HL: questo è il valore iniziale per il BIP. Poi in DE viene caricato 0020 (Hex), cioè di quanto HL verrà ridotto ogni volta: questo valore lo chiamiamo "passo". Poi si sottrae DE da HL per diminuire il valore di HL di un valore pari al "passo". In DE viene caricato un numero che determina quanto tempo il computer debba rimanere su ciascun tono prima di passare ad uno più alto. Prima di fare la chiamata alla routine della ROM, HL viene posto su uno stack per memorizzarne il valore. Dopo che la ROM ha svolto il suo compito, richiamiamo HL dallo stack e controlliamo il registro H per vedere se è giunto a zero: se sì, il programma termina, altrimenti passiamo al tono successivo. Tutti questi toni insieme danno come risultato un suono crescente di frequenza che può essere usato per i colpi di laser. Questa routine è molto versatile perciò provate ad utiizzare altri valori per il passo, per il valore

iniziale di HL e per la lungheza di ciascun tono. Facendo alcune prove si può ottenere una grande varietà di suoni, da un breve ZAP a suoni lunghi e crescenti. Può essere molto interessante provare a mettere tutti questi effetti in un ciclo in BASIC.

Come facciamo per i motori di un'astronave? Quel suono ha un nome particolare: "rumore" può sembrare non molto tecnico, ma è il termine usato per suoni come il fruscio di un registratore a nastro, che ha una frequenza elevata rispetto al basso rombo dei motori di un'astronave, che ha una frequenza bassa. Le persone che si interessano di sintetizzatori conosceranno già i diversi tipi di rumori: rosa, bianco, "heavy metal", etc.; qui ci limiteremo alle cose semplici, parlando soltanto di rumore.

Il rumore è in realtà una serie di impulsi ad intervalli casuali, quindi lo possiamo simulare abbastanza facilmente. Anzitutto abbiamo bisogno di una fonte di numeri casuali e invece di scrivere un qualche complicato generatore di numeri casuali useremo tutti i codici della ROM come dati casuali. Ciò significa che ci ripeteremo quando avremo esaurito quei codici, ma, dato che il rumore non è proprio molto melodico, sarà difficile notare la ripetizione. Il motore di una nave ha un rumore di bassa frequenza, quindi dovremo intervallare gli impulsi con un ciclo B. Useremo il registro HL per indicare a quale byte della ROM siamo arrivati e il registro DE per contare il numero di cicli voluto. Ecco come si presenta graficamente il nostro rumore:

ed ecco il programma che lo genera:

```
7000 F3       DI
7001 210000   LD    HL,0000
7004 110020   LD    DE,2000
7007 7E       LD    A,(HL)
7008 D3FE     OUT   (FE),A
700A 06FF     LD    B,FF
700C 10FE     DJNZ  700C
700E 1B       DEC   DE
700F 23       INC   HL
7010 7B       LD    A,E
7011 B2       OR    D
7012 20F3     JR    NZ,7007
7014 FB       EI
7015 C9       RET
```

Provate a far girare il programma usando il loader in BASIC. noterete che il programma produce anche ogni sorta di colori sulla cornice, il che può essere molto simpatico. Come si potrebbe fare se si volesse eliminare questo effetto? Suggerimento: usando AND.

Adesso come fare per le esplosioni? Dunque, la tecnica è abbastanza simile, usiamo lo stesso rumore ma lo facciamo andare da una frequenza elevata ad una bassa. Per far ciò allunghiamo l'intervallo fra ciascun impulso producendo un rumore che cala in frequenza.
Ecco il programma che fa ottenere questo effetto:

```
7000 210000   LD    HL,0000
7003 0E00     LD    C,00
7005 1620     LD    D,20
7007 7E       LD    A,(HL)
7008 E618     AND   18
700A D3FE     OUT   (FE),A
700C 41       LD    B,C
700D 10FE     DJNZ  700D
700F 23       INC   HL
7010 15       DEC   D
7011 20F4     JR    NZ,7007
7013 0C       INC   C
7014 20EF     JR    NZ,7005
7016 C9       RET
```

In questo programma usiamo C come un contatore che rende il ciclo B sempre più lungo. Possiamo rallentare tutto l'effetto aumentando il valore inziale di D. Se invece volete accelerarlo dovete ridurre il valore iniziale di D. Con un'esplosione di breve durata, richiamata ad intervalli casuali da un ciclo in BASIC, si può produrre l'effetto di tuoni e lampi. Cambiando l'istruzione INC C con DEC C (codice OD) si potrà ottenere un rumore di intensità crescente, che è molto efficace in un gioco per quando qualcosa compare sullo schermo (gli assi di video giochi conoscono bene questo effetto nei giochi Defender). In questo programma c'è anche la soluzione al mio precedente quesito, su come eliminare i vari colori della cornice: abbiamo usato AND 18 per far sì che solo i bit 3 e 4 vengono modificati.

Bene, questa è la fine del sesto capitolo e spero che essa vi abbia dato alcune idee per produrre effetti sonori e anche un'idea di come i computer riescano a comunicare. Una lista completa delle istruzioni IN e OUT viene riportata nell'Appendice D insieme alle definizioni. Se volete fare delle prove con altre porte oltre a quelle che già sono montate sullo Spectrum, dovrete acquistare una porta aggiuntiva. Ce ne sono diversi modelli disponibili, ma non è molto importante quale comprate purché vi rendiate conto che sono diverse: alcune hanno i canali di ingresso e di uscita distinti, altre usano circuiti integrati più complessi e sono

programmabili. In genere, per gli usi domestici, quanto più numerose sono le linee di ingresso e di uscita, tanto meglio. Inoltre, assicuratevi di conoscere bene le istruzioni IN e OUT del BASIC, perché se le conoscete bene sarete anche in grado di usare le istruzioni equivalenti in linguaggio macchina.

CAPITOLO 7

LE INTERRUZIONI

In questo capitolo vedremo gli "interrupt", o interruzioni, prima di procedere ad esaminare la ROM. Le interruzioni sono per voi di scarsa utilità diretta sullo Spectrum, però ci possono aiutare a capire un po' del funzionamento della macchina. Ad intervalli regolari in un computer ci sono certi compiti che la ROM deve eseguire.

Tali compiti variano da macchina a macchina: una fondamentale esigenza è quella di aggiornare una qualche specie di timer o di contatore. Un contatore consente di avere una scansione precisa del tempo, per esempio per un orologio. Il modo in cui il computer esegue questi vari compiti è mediante le interruzioni.

Un'interruzione è un segnale inviato alla CPU da un dispositivo esterno, ad esempio dal circuito logico. Esso serve per comunicare alla CPU di interrompere ciò che stava facendo e di iniziare ad eseguire un altro compito. Quando lo ha eseguito o, con termini più tecnici ha finito di gestire l'interruzione, essa riprende ciò che stava facendo prima dal punto in cui si era fermata.

Esistono due tipi di interruzione, un'interruzione "mascherabile" e un'interruzione "non mascherabile". Un'interruzione mascherabile si ha quando si può dire alla CPU di ignorarlo quando lo si desidera. Nello ZX Spectrum della Sinclair, un'interruzione mascherabile viene generata ogni 1/50 di secondo: essa viene generata dal circuito logico che invia un impulso al piedino chiamato INT della CPU Z80-A.

Quando l'impulso viene ricevuto, la subroutine posta all'indirizzo 0038 Hex viene chiamato in esecuzione. Eccone il programma:

Routine di interruzione

```
0038 F5       PUSH  AF
0039 E5       PUSH  HL
003A 2A785C   LD    HL,(5C78)
003D 23       INC   HL
003E 22785C   LD    (5C78),HL
0041 7C       LD    A,H
0042 B5       OR    L
0043 2003     JR    NZ,0048
0045 FD3440   INC   (IY+40)
0048 C5       PUSH  BC
0049 D5       PUSH  DE
004A CDBF02   CALL  02BF
004D D1       POP   DE
004E C1       POP   BC
004F E1       POP   HL
0050 F1       POP   AF
0051 FB       EI
0052 C9       RET
```

Anzitutto i registri AF e HL vengono posti sullo stack: in questo modo i valori in essi contenuti non vengono modificati, dato che, ricordate, una routine è stata appena interrotta e quando avremo finito di gestire l'interruzione quella routine dovrà essere continuata come se nulla fosse successo. Il passo seguente è caricare in HL i primi due byte, quelli meno significativi della variabile di sistema FRAMES. Questo numero a due byte viene poi incrementato di uno e si esegue un test per vedere se si è giunti a 0. Il test viene eseguito con la tecnica dell'istruzione "OR" vista nel capitolo 4: caricando H in A e operando con OR su L, se HL è nullo sarà anche l'accumulatore, se HL non è nullo allora si salta all'istruzione INC (IY+40) perché IY contiene sempre l'indirizzo di base 5C3A: in questo mdoo quando la ROM deve fare riferimento ad una delle variabili di sistema può servirsi di IY più uno spostamento invece che della coppia di registri HL. Per esempio se volessimo usare HL nel nostro programma dovremmo fare:

```
PUSH HL
LD HL, 5C3A
INC (HL)
POP HL
```

Invece di

INC (IY+40)

molto più semplice.

Successivamente, gli altri registri vengono memorizzati sullo stack prima di richiamare una subroutine che inizia una scansione della tastiera con la variabile di sistema K-SCAN e che poi determina quale è stato l'ultimo tasto premuto e lo memorizza in LAST K. I registri vengono poi ripristinati ai loro precedenti valori recuperandoli dallo stack. Infine, prima del ritorno si incontra l'istruzione EI.

Essa è complementare all'istruzione DI. Non abbiamo ancora incontrato queste istruzioni, quindi esaminiamole più da vicino.

La loro funzione è molto semplice. Ho già accennato al fatto che esistono due tipi di interruzione: mascherabili e non mascherabili. Lo Spectrum si serve di un'interruzione mascherabile per la routine di aggiornamento del timer e scansione della tastiera. Quando poi viene generata un'altra interruzione mascherabile, la CPU automaticamente la disattiva, cioè ignora ogni ulteriore interruzione mascherabile, in modo che un'interruzione non possa essere interrotta! Succede proprio come se si appendesse un avviso "Non Disturbare" alla porta. Quando ha finito di gestire l'interruzione il computer in pratica toglie l'avviso usando l'istruzione EI. "EI" sta per "Enable Interrupt", abilita le interruzioni. Si deve dare questa istruzione altrimenti le interruzioni verrebbero ignorate per sempre, e senza le interruzioni lo ZX Spectrum non saprebbe più se vengono premuti altri tasti.

Ci sono altre circostanze in cui la ROM non vuole essere disturbata. Per esempio quando sta compiendo un'operazione con il registratore a nastro: se per esempio il circuito logico invia un'interruzione durante la lettura di un byte, quando si è finito di aggiornare il time e di fare la scansione della tastiera quel byte può già essere stato superato sul nastro.

Quindi, per evitare interferenze, possiamo usare l'istruzione DI, che significa "Disable Interrupt", disabilita le interruzioni. Come già accennato, essa serve semplicemente per dire alla CPU di ignorare le interruzioni mascherabili fino ad un'ulteriore istruzione, per esempio EI. Detto per inciso, ogni volta che viene dato un reset alla CPU, le interruzioni mascherabili sono sempre abilitate.

I MODI DI INTERRUZIONE

Non vi ho ancora detto che la CPU non è obbligata ad andare alla subroutine all'indirizzo 0038 Hex quando si verifica un'interruzione: essa va lì perché si trova nel "modo di interruzione uno". Dunque cosa sono questi misteriosi "modi di interruzione"? La prima cosa da dire è che per un semplice proprietario di uno Spectrum essi sono praticamente irrilevanti, dato che gli altri modi sono utilizzabili soltanto se si hanno sufficienti conoscenze di hardware. Il motivo per cui faccio qui menzione di questi modi è che uno di essi, il modo numero due, utilizza il registro "I" il cui uso è per voi probabilmente un mistero.

Purtroppo a questo punto sono costretto a muovere qualche passo nel temibile mondo dell'hardware per spiegare i modi, quindi cercate di seguirmi e io mi sforzerò di spiegare tutto con linguaggio il più semplice possibile.

IL MODO DI INTERRUZIONE ZERO

Le interruzioni vengono spesso usate come il saluto "Ciao!" quando due computer stanno parlando tra loro. Ovviamente essi non parlano realmente; se inviano messaggi o in serie (allo stesso modo in cui un programma viene inviato al nastro magnetico), oppure in parallelo, per esempio un intero byte tutto in una volta. Prima di comunicare il mittente del messaggio deve attirare l'attenzione dell'altro. Ciò viene fatto per mezzo di un'interruzione (i computer poco amichevoli come il BBC e lo Spectrum, semplicemente si ignorano a vicenda mediante l'istruzione DI!). Ma cosa succede se c'è anche l'interruzione ogni 1/50 di secondo per cose come il timer? Ciò di cui abbiamo bisogno è che il computer si possa annunciare dicendo: "Ciao, sono l'altro computer, possiamo parlare?", cosicché la CPU non lo confonda con l'interruzione del timer. Una maniera per far ciò è per mezzo del modo di interruzione zero. In questo modo di interruzione si ha che il dispositivo che ha inviato un'interruzione, dopo averlo fatto, ha tempo sufficiente per trasmettere un byte alla CPU. Quando la CPU riceve quel byte essa lo esegue semplicemente come se fosse una qualsiasi altra istruzione. Ricordando che le istruzioni RST sono lunghe un byte, il computer può trasmettere il comando RST 28, per esempio. All'indirizzo 0038 Hex ci potrà essere una routine che riceve il messaggio proveniente dall'altro computer.
Analogamente, quando il circuito logico invia un'interruzione, volendo dire che è ora di incrementare il timer, esso può trasmettere il comando RST 38, cosicché la CPU passa all'indirizzo 0038 Hex per iniziare la routine di aggiornamento del timer. Quindi usando queste istruzioni RST in modo zero, un qualsiasi dispositivo può dire alla CPU quale routine eseguire.

IL MODO DI INTERRUZIONE UNO

Questo è il modo di interruzione che viene fissato non appena il computer viene acceso. Quando viene generata un'interruzione, si effettua una chiamata alla subroutine all'indirizzo 0038 Hex. Lo Spectrum utilizza questo modo di interruzione come abbiamo descritto prima.

IL MODO DI INTERRUZIONE DUE

Qui è dove entra in gioco il registro I. Il suo nome completo è "vettore di interruzioni", ovvero registro IV. Vettore è un termine tecnico usato per qualcosa che serve da puntatore in una tabella. Per ciascun dispositivo che può generare un'interruzione c'è un certo indirizzo di una subroutine memorizzato come parte di una tabella nella memoria.

Tabella di interrupt

8000	10	00
8002	00	70
.	.	.
.	.	.
.	.	.

Per sapere quale subroutine chiamare, quando la CPU riceve un'interruzione viene generato un puntatore che viene posto su due byte di dati nella tabella. Questo puntatore è formato prendendo il registro I come byte più significativo e il byte che il dispositivo invia alla CPU come il byte meno significativo. Quindi, facendo riferimento alla tabella di interruzione, se per esempio il registro I contiene 80 e il dispositivo invia 00, verrà chiamata la subroutine che inizia all'indirizzo 00 10. Vediamo i vari passi per chiarire il concetto:

1. viene generata un'interruzione;
2. la CPU riceve un byte dal dispositivo mittente;
3. usando il registro I come byte più significativo ed il byte ricevuto come il meno significativo, viene formato un vettore (puntatore);
4. all'indirizzo cui punta questo vettore c'è un byte meno significativo seguito dal byte più significativo dell'indirizzo di una subroutine;
5. questi due byte vengono usati per una chiamata ad una subroutine.

Per cambiare il modo di interruzione possiamo usare le istruzioni IM0, IM1.IM2. Quando la macchina viene accesa, la ROM esegue una routine che inizializza tutto e che, prima di abilitare l'interruzione, esegue un'istruzione IM1. Ecco i codici esadecimali per le istruzioni con cui fissare il modo di interruzione:

IM0	**ED 46**
IM1	**ED 56**
IM2	**ED 5E**

INTERRUZIONI NON MASCHERABILI

Come ho già accennato, esistono due tipi di interruzione. L'interruzione non mascherabile (NMI), è quella che ora considereremo. Essendo non mascherabile, l'unica circostanza in cui il computer la ignora è durante la gestione di un'altra interruzione. Quando un NMI viene generato, il computer passa ad una subroutine all'indirizzo 0066 Hex niente se, niente ma, niente modi, semplicemente 0066/ Hex. Sfortunatamente però la subroutine all'indirizzo 0066Hex sembra contenere un errore. Ecco il listato:

```
0066 F5         PUSH AF
0067 E5         PUSH HL
0068 2AB05C     LD   HL,(5C80)
006B 7C         LD   A,H
006C B5         OR   L
006D 2001       JR   NZ,0070
006F E9         JP   HL
0070 E1         POP  HL
0071 F1         POP  AF
0072 ED45       RETN
```

La prima cosa da far notare è che la routine viene terminata con il comando RETN: questo viene usato al posto di RET, perché si tratta di una routine di gestione di un'interruzione non mascherabile. Il punto più importante è però l'errore. Studiando il programma, notiamo che anzitutto le coppie AF e HL vengono poste sullo stack. Poi i contenuti degli indirizzi 5C80 e 5C81 vengono caricati in HL; se cercate cosa siano questi due indirizzi, troverete che si tratta degli unici due byte liberi fra le variabili di sistema. A questo punto la trama si complica: un test di azzeramento con "OR" viene eseguito su HL usando il metodo che ho descritto nel capitolo quattro. L'istruzione successiva avrebbe dovuto essere JR Z, 0070, invece è JR NZ, 0070. Il fatto è che se fosse stata JR Z e quei due byte non fossero nulli, si effettuerebbe un salto all'indirizzo in HL. In questo modo si sarebbe potuto avere un tasto collegato al NMI, e caricando in quei due byte l'indirizzo di una routine appropriata si avrebbe una possibliità di recupero da una condizione di stallo, ad esempio un ciclo infinito.

Ma così com'è, si effettuerà il salto soltanto se quei due byte sono zero, il che opera come una funzione quasi totalmente inutile di reset totale. Se non fosse stato commesso questo errore, sarebbe stata aperta la strada per avere tasti funzione programmabili e molte altre applicazioni. Pazienza, avremo più fortuna la prossima volta...

APPENDICI

A: Tabella di raffronto dei codici esadecimali, decimali e ASCII, comprensiva delle mnemoniche per lo Z80.

B: Le mnemoniche per lo Z80 ed i segnali.

C: Le variabili di sistema con spiegazione.

D: Le mnemoniche per lo Z80 con brevi definizioni.

I CODICI ESADECIMALI E DECIMALI ED I CARATTERI ASCII. LE MNEMONICHE PER LO Z80

Questa appendice può servire come una completa tabella di raffronto che comprende i numeri esadecimali da 00 a FF, i loro equivalenti decimali, i caratteri ASCII dello Spectrum ZX e, infine, non meno importanti, le mnemoniche del linguaggio macchina dello Z80. Questi vengono elencati in tre sezioni: senza prefisso, con il prefisso CB e con il prefisso ED. Quelle istruzioni che operano con i registri indice IX e IY hanno dei codici esadecimali che sono semplicemente quelli delle istruzioni equivalenti ad HL prefissati da DD per IX e da FD per IY.

ESA	CARATTERE DEC.		Z80 SENZA PREFISSO	PREFISSO CB	PREFISSO ED
00	0	Non utilizzati	NOP	RLC B	
01	1		LD BC,nn	RLC D	
02	2		LD (BC),A	RLC D	
03	3		INC BC	RLC E	
04	4		INC B	RLC H	
05	5		BEC B	RLC L	
06	6	**PRINT** comma	LD B,n	RLC (HL)	
07	7	**EDIT**	RLCA	RLC A	
08	8	cursore sinistra	EX AF,AF'	RRC B	
09	9	cursore destra	ADD HL,BC	RRC C	
0A	10	cursore giù	LD A,(BC)	RRC D	
0B	11	cursore su	DEC BC	RRC E	
0C	12	**DELETE**	INC C	RRCH	
0D	13	**ENTER**	DEC C	RRC L	
0E	14	numero	LD C,n	RRC (HL)	
0F	15	non utilizzato	RRCA	RRC A	
10	16	**INK** controllo	DJNZ dis	RL B	

ESA	CARATTERE	DEC.	Z80 SENZA PREFISSO	PREFISSO CB	PREFISSO ED
11	17	controllo **PAPER**	LD DE,nn	RL C	
12	18	controllo **FLASH**	LD (DE),A	RL D	
13	19	controllo **BRIGHT**	INC DE	RL E	
14	20	controllo **INVERSE**	INC D	RL H	
15	21	controllo **OVER**	DEC D	RL L	
16	22	controllo **AT**	LD D,n	RL (HL)	
17	23	controllo **TAB**	RLA	RLA	
18	24		JR dis	RR B	
19	25		ADD HL,DE	RR C	
1A	26		LD A,(DE)	RR D	
1B	27	non utilizzati	DEC DE	RR E	
1C	28		INC E	RR H	
1D	29		DEC E	RR L	
1E	30		LD E,n	RR (HL)	
1F	31		RRA	RR A	
20	32	spazio	JR NZ,dis	SLA B	
21	33	!	LD HL,nn	SLA C	
22	34	“	LD (nn),HL	SLA D	
23	35	#	INC HL	SLA E	
24	36	$	INC H	SLA H	
25	37	%	DEC H	SLA L	
26	38		LD H,n	SLA (HL)	
27	39	'	DAA	SLA A	
28	40	(	JR Z,dis	SRA B	
29	41	)	ADD HL,HL	SRA C	
2A	42	*	LD HL,(nn)	SRA D	
2B	43	+	DEC HL	SRA E	
2C	44	,	INC L	SRA H	
2D	45	–	DEC L	SRA L	
2E	46	.	LD L,n	SRA (HL)	
2F	47	/	CPL	SRA A	
30	48	0	JR NC,dis		
31	49	1	LD SP,nn		
32	50	2	LD (nn),A		
33	51	3	INC SP		
34	52	4	INC (HL)		
35	53	5	DEC (HL)		
36	54	6	LD (HL),n		
37	55	7	SCF		
38	56	8	JR C,dis	SRL B	
39	57	9	ADD HL,SP	SRL C	
3A	58	:	LD A,(nn)	SRL D	
3B	59	;	DEC SP	SRL E	
3C	60	<	INC A	SRL H	
3D	61	=	DEC A	SRL L	
3E	62	>	LD A,n	SRL (HL)	
3F	63	?	CCF	SRL A	

ESA	CARATTERE	DEC.	Z80 SENZA PREFISSO	PREFISSO CB	PREFISSO ED
40	64	@	LD B,B	BIT 0,B	IN B,(C)
41	65	A	LD B,C	BIT 0,C	OUT (C),B
42	66	B	LD B,D	BIT 0,D	SBC HL,BC
43	67	C	LD B,E	BIT 0,E	LD (nn),BC
44	68	D	LD B,H	BIT 0,H	NEG
45	69	E	LD, B,L	BIT 0,1	RETN
46	70	F	LD B,(HL)	BIT 0,(HL)	IM 0
47	71	G	LD B,A	BIT 0,A	LD I,A.
48	72	H	LD C,B	BIT 1,B	IN C,(C)
49	73	I	LD C,C	BIT 1,C	OUT (C),C
4A	74	J	LD C,D	BIT 1,D	ADC HL,BC
4B	75	K	LD C,E	BIT 1,E	LD BC,(nn)
4C	76	L	LD C,H	BIT 1,H	
4D	77	M	LD C,L	BIT 1,L	RETI
4E	78	N	LD C,(HL)	BIT 1,(HL)	
4F	79	O	LD C,A	BIT 1,A	LD R,A
50	80	P	LD D,B	BIT 2,B	IN D,(C)
51	81	Q	LD D,C	BIT 2,C	OUT (C),D
52	82	R	LD D,D	BIT 2,D	SBC HL,DE
53	83	S	LD D,E	BIT 2,E	LD (nn),DE
54	84	T	LD D,H	BIT 2,H	
55	85	U	LD D,L	BIT 2,L	
56	86	V	LD D,(HL)	BIT 2,(HL)	IM 1
57	87	W	LD D,A	BIT 2,A	LD A,I
58	88	X	LD E,B	BIT 3,B	IN E,(C)
59	89	Y	LD E,C	BIT 3,C	OUT (C),E
5A	90	Z	LD E,D	BIT 3,D	ADC HL,DE
5B	91	[	LD E,E	BIT 3,E	LD DE,(nn)
5C	92	/	LD E,H	BIT 3,H	
5D	93	]	LD E,L	BIT 3,L	
5E	94	↑	LD E,(HL)	BIT 3,(HL)	IM 2
5F	95	–	LD E,A	BIT 3,A	LD A,R
60	96	£	LD H,B	BIT 4,B	IN H,(C)
61	97	a	LD H,C	BIT 4,C	OUT (C),H
62	98	b	LD H,D	BIT 4,D	SBC HL,HL
63	99	c	LD H,E	BIT 4,E	LD (nn),HL
64	100	d	LD H,H	BIT 4,H	
65	101	e	LD H,L	BIT 4,L	
66	102	f	LD H,(HL)	BIT 4,(HL)	
67	103	g	LD H,A	BIT 4,A	RRD
68	104	h	LD L,B	BIT 5,B	IN L,(C)
69	105	i	LD L,C	BIT 5,C	OUT (C),L
6A	106	j	LD L,D	BIT 5,D	ADC HL,HL
6B	107	k	LD L,E	BIT 5,E	LD HL,(nn)
6C	108	l	LD L,H	BIT 5,H	
6D	109	m	LD, L,L	BIT 5,L	
6E	110	n	LD L,(HL)	BIT 5,(HL)	
6F	111	o	LD L,A	BIT 5,A	RLD
70	112	p	LD (HL),B	BIT 6,B	IN F,(C)
71	113	q			
LD (HL),C	BIT 6,C				
72	114	r	LD (HL),D	BIT 6,D	SBC HL,SP
73	115	s	LD (HL),E	BIT 6,E	LD (nn),SP
74	116	t	LD (HL),H	BIT 6,H	

ESA	CARATTERE	DEC.	Z80 SENZA PREFISSO	PREFISSO CB	PREFISSO ED
75	117	u	LD (HL),L	BIT 6,L	
76	118	v	HALT	BIT 6,(HL)	
77	119	w	LD (HL),A	BIT 6,A	
78	120	x	LD A,B	BIT 7,B	IN A,(C)
79	121	y	LD A,C	BIT 7,C	OUT (C),A
7A	122	z	LD A,D	BIT 7,D	ADC HL,SP
7B	123	{	LD A,E	BIT 7,E	LD SP,(nn)
7C	124	|	LD A,H	BIT 7,H	
7D	125	}	LD A,L	BIT 7,L	
7E	126	-	LD A,(HL)	BIT 7,(HL)	
7F	127	©	LD A,A	BIT 7,A	
80	128	■	ADD A,B	RES 0,B	
81	129	■	ADD A,C	RES 0,C	
82	130	■	ADD A,D	RES 0,D	
83	131	■	ADD A,E	RES 0,E	
84	132	■	ADD A,H	RES 0,H	
85	133	■	ADD A,L	RES 0,L	
86	134	■	ADD A,(HL)	RES 0,(HL)	GRAPHICS
87	135	■	ADD A,A	RES 0,A	
88	136	■	ADC A,B	RES 1,B	
89	137	■	ADC A,C	RES 1,C	
8A	138	■	ADC A,D	RES 1,D	
8B	139	■	ADC A,E	RES 1,E	
8C	140	■	ADC A,H	RES 1,H	
8D	141	■	ADC A,L	RES 1,L	
8E	142	■	ADC A,(HL)	RES 1,(HL)	
8F	143	■	ADC A,A	RES 1,A	
90	144	(a)	SUB B	RES 2,B	
91	145	(b)	SUB C	RES 2,C	
92	146	(c)	SUB D	RES 2,D	
93	147	(d)	SUB E	RES 2,E	
94	148	(e)	SUB H	RES 2,H	
95	149	(f)	SUB L	RES 2,L	
96	150	(g)	SUB (HL)	RES 2,(HL)	
97	151	(h)	SUB A	RES 2,A	
98	152	(i)	SBC A,B	RES 3,B	
99	153	(j)	SBC A,C	RES 3,C	
9A	154	(k) user	SBC A,D	RES 3,D	
9B	155	(l) graphics	SBC A,E	RES 3,E	
9C	156	(m)	SBC A,H	RES 3,H	
9D	157	(n)	SBC A,L	RES 3,L	
9E	158	(o)	SBC A,(HL)	RES 3,(HL)	
9F	159	(p)	SBC A,A	RES 3,A	
A0	160	(q)	AND B	RES 4,B	LDI
A1	161	(r)	AND C	RES 4,C	CPI
A2	162	(s)	AND D	RES 4,D	INI
A3	163	(t)	AND E	RES 4,E	OUTI
A4	164	(u)	AND H	RES 4,H	
A5	165	**RND**	AND L	RES 4,L	
A6	166	**INKEY$**	AND (HL)	RES 4,(HL)	
A7	167	**PL**	AND A	RES 4,A	
A8	168	**FN**	XOR B	RES 5,B	LDD
A9	169	**POINT**	XOR C	RES 5,C	CPD
AA	170	**SCREEN$**	XOR D	RES 5,D	IND

ESA	CARATTERE	DEC.	Z80 SENZA PREFISSO	PREFISSO CB	PREFISSO ED
AB	171	**ATTR**	XOR E	RES 5,E	OUTD
AC	172	**AT**	XOR H	RES 5,H	
AD	173	**TAB**	XOR L	RES 5,L	
AE	174	**VAL$**	XOR (HL)	RES 5,(HL)	
AF	175	**CODE**	XOR A	RES 5,A	
B0	176	**VAL**	OR B	RES 6,B	LDIR
B1	177	**LEN**	OR C	RES 6,C	CPIR
B2	178	**SIN**	OR D	RES 6,D	INIR
B3	179	**COS**	OR E	RES 6,E	OTIR
B4	180	**TAN**	OR H	RES 6,H	
B5	181	**ASN**	OR L	RES 6,L	
B6	182	**ACS**	OR (HL)	RES 6,(HL)	
B7	183	**ATN**	OR A	RES 6,A	
B8	184	**LN**	CP B	RES 7,B	LDDR
B9	185	**EXP**	CP C	RES 7,C	CPDR
BA	186	**INT**	CP D	RES 7,D	INDR
BB	187	**SQR**	CP E	RES 7,E	OTDR
BC	188	**SGN**	CP H	RES 7,H	
BD	189	**ABS**	CP L	RES 7,L	
BE	190	**PEEK**	CP (HL)	RES 7,(HL)	
BF	191	**IN**	CP A	RES 7,A	
C0	192	**USR**	RET NZ	SET 0,B	
C1	193	**STR$**	POP BC	SET 0,C	
C2	194	**CHR$**	JP NZ,nn	SET 0,D	
C3	195	**NOT**	JP nn	SET 0,E	
C4	196	**BIN**	CALL NZ,nn	SET 0,H	
C5	197	**OR**	PUSH BC	SET 0,L	
C6	198	**AND**	ADD A,n	SET 0,(HL)	
C7	199	<=	RST 0	SET 0,A	
C8	200	>=	RET Z	SET 1,B	
C9	201	<>	RET	SET 1,C	
CA	202	**LINE**	JPZ,nn	SET 1,D	
CB	203	**THEN**		SET 1,E	
CC	204	**TO**	CALLZ,nn	SET 1,H	
CD	205	**STEP**	CALL nn	SET 1,L	
CE	206	**DEF FN**	ADC A,n	SET 1,(HL)	
CF	207	**CAT**	RST 8	SET 1,A	
D0	208	**FORMAT**	RET NC	SET 3,B	
D1	209	**MOVE**	POP DE	SET 2,C	
D2	210	**ERASE**	JP NC,nn	SET 2,D	
D3	211	**OPEN #**	OUT (n),A	SET 2,E	
D4	212	**CLOSE #**	CALL NC,nn	SET 2,H	
D5	213	**MERGE**	PUSH DE	SET 2,L	
D6	214	**VERIFY**	SUB n	SET 2,(HL)	
D7	215	**BEEP**	RST 16	SET 2,A	
D8	216	**CIRCLE**	RET C	SET 3,B	
D9	217	**INK**	EXX	SET 3,C	
DA	218	**PAPER**	JP C,nn	SET 3,D	
DB	219	**FLASH**	IN A,(n)	SET 3,E	
DC	220	**BRIGHT**	CALL C,nn	SET 3,H	
DD	221	**INVERSE**	PREFISSA LE ISTRUZIONI CHE USANO IX	SET 3,L	
DE	222	**OVER**	SBC A,n	SET 3,(HL)	

ESA	CARATTERE	DEC.	Z80 SENZA PREFISSO	PREFISSO CB	PREFISSO ED
DF	223	**OUT**	RST 24	SET 3,A	
E0	224	**LPRINT**	RET PO	SET 4,B	
E1	225	**LLIST**	POP HL	SET 4,C	
E2	226	**STOP**	JP PO,nn	SET 4,D	
E3	227	**READ**	EX (SP),HL	SET 4,E	
E4	228	**DATA**	CALL PO,nn	SET 4,H	
E5	229	**RESTORE**	PUSH HL	SET 4,L	
E6	230	**NEW**	AND n	SET 4,(HL)	
E7	231	**BORDER**	RST 32	SET 4,A	
E7	232	**CONTINUE**	RET PE	SET 5,B	
E9	233	**DIM**	JP (HL)	SET 5,C	
EA	234	**REM**	JP PE,nn	SET 5,D	
EB	235	**FOR**	EX DE,HL	SET 5,E	
EC	236	**GO TO**	CALL PE,nn	SET 5,H	
ED	237	**GO SUB**		SET 5,L	
EE	238	**INPUT**	XOR n	SET 5,(HL)	
EF	239	**LOAD**	RST 40	SET 5,A	
F0	240	**LIST**	RET P	SET 6,B	
F1	241	**LET**	POP AF	SET 6,C	
F2	242	**PAUSE**	JP P,nn	SET 6,D	
F3	243	**NEXT**	DI	SET 6,E	
F4	244	**POKE**	CALL P,nn	SET 6,H	
F5	245	**PRINT**	PUSH AF	SET 6,L	
F6	246	**PLOT**	OR n	SET 6,(HL)	
F7	247	**RUN**	TST 48	SET 6,A	
F8	248	**SAVE**	RET M	SET 7,B	
F9	249	**RANDOMIZE**	LD SP,HL	SET 7,C	
FA	250	**IF**	JP M,nn	SET 7,D	
FB	251	**CLS**	EI	SET 7,E	
FC	252	**DRAW**	CALL M,nn	SET 7,H	
FD	253	**CLEAR**	PREFISSA LE ISTRUZIONI CHE USANO IY	SET 7,L	
FE	254	**RETURN**	CP n	SET 7,(HL)	
FF	255	**COPY**	RST 56	SET 7,A	

APPENDICE B

I SEGNALI

In questa appendice vengono elencate le varie istruzioni con a fianco di ciascuna l'indicazione del suo effetto sui segnali. Soltanto i segnali più importanti vengono indicati: il segnale di riporto, il segnale di parità/overflow, il flag di zero e il flag di segno. Gli altri segnali non sono molto importanti per il programmatore dato che non giocano alcun ruolo nel prendere decisioni mediante le istruzioni JR, JP, CALL e RET. Nella tabella vengono usati i seguenti simboli:

Per le mnemoniche:	nn	Un byte singolo di dati
	nnnn	Due byte di dati
	r	Un registro ad un solo byte (A,B,-C,D,E,H,L)
	d	Un registro da due byte
	c	Una condizione
	dis	(Abbreviazione di "displacement") il dato per uno spostamento relativo, espresso come complemento a due.
Per i segnali:	0	Il segnale viene posto a zero
	1	Il segnale viene posto a uno
	R	Il segnale viene modificato a seconda del risultato
	?	Il segnale viene posto a uno o a zero in modo casuale
	B	Il segnale viene posto a uno se il registro B o la copia BC (secondo tipo di istruzione) risulta nullo alla fine dell'operazione.

ISTRUZIONI mnemoniche	FLAG			
	S	Z	P	C
ADC A,r	R	R	R	R
ADC HL,d	R	R	R	R
ADD A,r	R	R	R	R
ADD HL,d	.	.	.	R
ADD IX,d	.	.	.	R
ADD IY,d	.	.	.	R
AND r	R	R	R	0
BIT b,r	?	R	?	.
CALL nnnn	.	.	.	.
CALL c,nnnn	.	.	.	.
CCF	.	.	.	R
CP r	R	R	R	R
CPI	R	B	R	.
CPD	R	B	R	.
CPDR	R	B	R	.
CPL	R	B	R	.
DAA	.	.	.	.
DEC r	R	R	R	R
DEC d	R	R	R	.
DI	.	.	.	.
DJNZ dis	.	.	.	.
EI	.	B	.	.
EX AF,AF'	.	.	.	.
EX DE,HL	.	.	.	.
EX (SP),HL	.	.	.	.
EX (SP),IX	.	.	.	.
EX (SP),IY	.	.	.	.
EXX	.	.	.	.
HALT	.	.	.	.
IM 0	.	.	.	.
IM 1	.	.	.	.
IM 2	.	.	.	.
INC r	.	.	.	.
INC d	R	R	R	.
IN A,(nn)	.	.	.	.
IN r,(C)	R	R	R	.
INI	?	B	?	.
IND	?	B	?	.
INIR	?	1	?	.
INDR	?	1	?	.
JP nnnn	.	.	.	.

ISTRUZIONI mnemoniche	FLAG			
	S	Z	P	C
JP c,nnnn	.	.	.	.
JP (HL)	.	.	.	.
JP (IX)	.	.	.	.
JP (IY)	.	.	.	.
JR dis	.	.	.	.
JR c,dis	.	.	.	.
LD (d),A	.	.	.	.
LD A,(d)	.	.	.	.
LD A,R	R	R	R	.
LD A,I	R	R	R	.
LD I,A	.	.	.	.
LD R,A	.	.	.	.
LD SP,IX	.	.	.	.
LD SP,IY	.	.	.	.
LD r,r	.	.	.	.
LD r,nn	.	.	.	.
LD d,nnnn	.	.	.	.
LD (nnnn),A	.	.	.	.
LD d,(nnnn)	.	.	.	.
LD (nnnn),d	.	.	.	.
LDI	.	.	B	.
LDD	.	.	B	.
LDIR	.	.	Ø	.
LDDR	.	.	Ø	.
NEG	R	R	R	R
NOP	.	.	.	.
OR r	R	R	R	Ø
OUR (nn),A	.	.	.	.
OUT (C),r	.	.	.	.
OUTI	?	B	?	.
OUTD	?	B	?	.
OTIR	?	1	?	.
OTDR	?	1	?	.
POP AF	R	R	R	R
POP d	.	.	.	.
PUSH AF	.	.	.	.
PUSH d	.	.	.	.
RES b,r	.	.	.	.
RET	.	.	.	.
RET c	.	.	.	.
RETN	.	.	.	.

ISTRUZIONI mnemoniche	FLAG S	Z	P	C
RETI	.	.	.	.
RLA	.	.	.	R
RLCA	.	.	.	R
RRA	.	.	.	R
RRCA	.	.	.	R
RL r	R	R	R	R
RLC r	R	R	R	R
RR r	R	R	R	R
RRC r	R	R	R	R
RRD	R	R	R	.
RST 00	.	.	.	.
RST 08	.	.	.	.
RST 10	.	.	.	.
RST 18	.	.	.	.
RST 20	.	.	.	.
RST 28	.	.	.	.
RST 30	.	.	.	.
RST 38	.	.	.	.
SBC A,r	R	R	R	R
SBC HL,d	R	R	R	R
SCF	.	.	.	1
SET b,r	.	.	.	.
SLA r	R	R	R	R
SRA r	R	R	R	R
SRL r	R	R	R	R
SUB r	R	R	R	R
XOR r	R	R	R	0

N.B. Nei casi in cui venga indicato “r” nella mnemonica, esso fa riferimento non solo ai registri ad un solo byte, ma anche a (HL), (IX + dis), (IY + dis) o semplicemente ad una costante numerica “nn”, qualora ciò sia previsto per la relativa istruzione.

APPENDICE C

LE VARIABILI DI SISTEMA

Questa appendice elenca tutte le variabili di sistema e ne dà una breve spiegazione, spesso in modo più esteso che nel manuale dello Spectrum. L'appendice può risultare più chiara a coloro che hanno una certa conoscenza del meccanismo di funzionamento della ROM. È consigliabile che le variabili contrassegnate da una "X" vengano modificate soltanto se se ne comprende bene l'effetto.

n° di byte	*Indirizzo esadecimale*	*Nome*	*Funzione*
8	5C00	KSTATE	Questa variabile consta di 8 byte. Ciascun byte contiene delle informazioni riguardo l'ultimo tasto premuto, per esempio quando deve essere ripetuto e il suo codice in modo esteso.
1	5C08	LAST K	Questa viene posta ad indicare l'ultimo tasto premuto, a seconda del modo. Essa viene modificata soltanto quando si preme un altro tasto. La ripetizione automatica funziona mediante questa variabile. Azzerandola e sottoponendola ad un test, si può rimanere in attesa della pressione di un qualsiasi tasto.
1	5C09	REPDEL	Indica per quanto tempo un tasto deve essere tenuto premuto prima che inizi a ripetersi. Il tempo viene espresso in 1/50 di secondo, oppure in 1/60 di secondo in Nord America. Il valore iniziale è 32 Hex.
1	5C0A	REPPER	L'intervallo di tempo (in 1/50 di secondo, o 1/60 in Nord America/ fra ogni successiva ripetizione di un tasto. Valore iniziale: 05 Hex. Lo si può ricondurre fino ad un minimo di 01 Hex, per aumentare la velocità della ripetizione.

n° di byte	Indirizzo esadecimale	Nome	Funzione
2	5C0B	DEFADD	L'indirizzo degli argomenti di una funzione definita dall'utente, quando se ne stia facendo uso.
1	5C0D	K DATA	Quando un codice di controllo del colore viene immesso direttamente dalla tastiera, per esempio extended shift–1 (inchiostro : blu), il secondo byte, per esempio il colore oppure il codice per il lampeggiamento etc., viene memorizzato in questa variabile mentre vengono inviati alla stampa i codici per INK, PAPER, BRIGHT, FLASH o INVERSE. Dopodiché la ROM richiama il secondo byte da questa variabile in modo da poterlo inviare alla stampa di seguito al codice di controllo del colore.
2	5C0E	TVDATA	Questa variabile viene usata dalla routine di stampa per memorizzare i controlli AT, TAB ed i colori da inviare al televisore.
X38	5C10	STRMS	Questa viene usata per memorizzare gli indirizzi di "offset" per la variabile CHANS. Per ognuno dei 16 file per l'utente e dei 3 file per il sistema c'è un indirizzo di offset: quando lo si somma al valore contenuto in CHANS esso punta all'inizio della routine di gestione di quel determinato file.
2	5C36	CHARS	In questa variabile è contenuto un valore pari a 100 Hex in meno dell'indirizzo dell'insieme dei caratteri (dallo spazio al simbolo di copyright). Normalmente il valore è 4C00 Hex (insieme dei caratteri all'indirizzo 4D00 Hex), ma lo si può modificare per puntare ad un insieme di caratteri definito dall'utente.
1	5C38	RASP	Il computer "gracchia" se vengono immessi i codici di controllo del colore in modo da produrre un colore non previsto. Esso gracchia anche se la linea di edit supera le 23 linee. Per cambiare questa lunghezza basta modificare il valore di questa variabile.

n° di byte	Indirizzo esadecimale	Nome	Funzione
1	5C39	PIP	La durata del "bip" della tastiera. Un valore più elevato di PIP fa sì che il suono sia più forte.
1	5C3A	ERR NR	Ha un valore inferiore di uno al codice di riporto. All'inizio vale FF Hex. Se in questa variabile si mette un valore con POKE dal BASIC e poi il programma termina l'esecuzione, quel valore viene stampato sul video come codice di errore.
X1	5C3B	FLAGS	I segnali di controllo del sistema BASIC.
X1	5C3C	TVFLAG	I segnali relativi al video e alla stampa.
X2	5C3D	ERR SP	Questa variabile punta ad un elemento nello stack della macchina. Quando si verifica un errore, questo elemento è l'indirizzo al quale si salta dopo che lo stack è stata reinizializzata dal comando RST Ø8. Modificando questo elemento potranno essere scritte nuove routine di gestione degli errori.
2	5C3F	LIST SP	Questa punta all'indirizzo di ritorno sullo stack della macchina dove si salta dopo aver effettuato un listato automatico.
1	5C41	MODE	Specifica il tipo di cursore: K,L,C,E, o G.
2	5C42	NEWPPC	La linea a cui si deve saltare. Usato con i GOTO e i GOSUB.
1	5C44	NSPPC	Il numero di istruzione nella linea cui si deve saltare. Mettendo un valore con POKE prima in NEWPCC, poi in NSPPC si può costringere il computer a saltare ad una specifica istruzione in una linea.
2	5C45	PPC	Numero di linea dell'istruzione in corso di esecuzione.
1	5C47	SUBPPC	Questo è il byte di attributi per la metà inferiore dello schermo. I bit da zero a due sono per il

nº di byte	Indirizzo esadecimale	Nome	Funzione
			colore INK e i bit da tre a cinque sono per il colore PAPER/BORDER. I bit FLASH e BRIGHT non sono utilizzati.
1	5C48	BORDCR	Numero della linea contenente il cursore del programma.
2	5C49	E PPC	Questa punta all'inizio della zona di memoria in cui sono tenute le variabili di un programma.
X2	5C4B	VARS	Indirizzo della variabile in assegnazione.
2	5C4D	DEST	Punta alla tabella degli indirizzi per la gestione dei file. Viene usata da STRMS.
X2	5C4F	CHANS	Punta all'indirizzo (nella tabella degli indirizzi per la gestione dei file) che viene attualmente utilizzato per la routine di gestione dei file.
X2	5C51	CURCHL	L'indirizzo del programma in BASIC.
X2	5C53	PROG	L'indirizzo della linea seguente nel programma in BASIC.
X2	5C55	NXTLIN	Punta al simbolo termine dell'ultima voce di DATA. Se nel programma non ci sono istruzioni DATA, questa variabile punta all'80 Hex alla fine del canale dei dati.
X2	5C57	DATADD	L'indirizzo del comando in corso di immissione.
X2	5C59	E LINE	L'indirizzo del cursore nella linea del comando.
2	5C5B	K CUR	L'indirizzo del successivo carattere da interpretare.
X2	5C5D	CHADD	L'indirizzo del carattere dopo il segno "?".
2	5C5F	X PTR	L'indirizzo del "work space" (zona di lavoro) temporaneo.

n° di byte	Indirizzo esadecimale	Nome	Funzione
X2	5C61	WORKSP	L'indirizzo della fine dello stack del calcolatore.
X2	5C63	STKBOT	L'indirizzo di inizio dello spazio libero.
X2	5C65	STKEND	Un registro del calcolatore usato per una serie di scopi relativi al calcolo.
1	5C67	BREG	L'indirizzo dell'area usata per le sei memorie del calcolatore (di solito, ma non sempre MEMBOT).
2	5C68	MEM	Altri segnali.
1	5C6A	FLAGS2	Il numero di linee (compresa una linea bianca) nella parte inferiore dello schermo.
X1	5C6B	DF SZ	Il numero della linea iniziale nel listato automatico.
2	5C6C	S TOP	Il numero della linea cui si salta con un CONTINUE.
2	5C6E	OLDPPC	Numero di linea a cui salta CONTINUE.
1	5C72	STRLEN	La lunghezza della destinazione del tipo di stringa in una assegnazione.
2	5C74	T ADDR	L'indirizzo dell'elemento successivo nella tabella di sintassi. Nella ROM c'è una grande tabella che riporta l'indirizzo delle routine per ciascun comando e definisce le modalità di reperimento delle informazioni necessarie.
2	5C76	SEED	Il seme per RND. Questa è la variabile che viene assegnata mediante l'istruzione RANDOMIZE.
3	5C78	FRAMES	Un contatore di quadro a tre byte che viene incrementato ogni 1/50 di secondi oppure ogni 1/60 in Nord America. Si veda il Capitolo 18 del manuale Sinclair.
2	5C7B	UDG	L'indirizzo del primo simbolo grafico definito dal-

nº di byte	Indirizzo esadecimale	Nome	Funzione
			l'utente. Ricordate che quando viene spostato verso il basso il RAMTOP per il linguaggio macchina, il valore di UDG non viene modificato. Quindi se si mette del linguaggio macchina nell'area UDG, si possono danneggiare i simboli già definiti.
1	5C7D	COORDS	Usata per memorizzare temporaneamente la coordinata X mentre vengono eseguiti i calcoli per il disegno.
1	5C7E		Come sopra ma per la coordinata Y.
1	5C7F	P POSN	Il numero di colonna (max 33) della posizione di stampa.
1	5C80	PR CC	Il byte meno significativo dell'indirizzo della posizione successiva per il comando LPRINT AT (nel buffer della stampante).
1	5C81		Non utilizzato.
2	5C82	ECHO E	Il numero di colonna (max 33) e numero di linea (nella metà inferiore - max 24) della fine del buffer di input.
2	5C84	DF CC	L'indirizzo della posizione PRINT per la "fetta" superiore del carattere nel file del video. Può essere modificato.
2	5C86	DFCCL	Come DF CC, ma per la parte inferiore del video.
X1	5C88	S POSN	Il numero di colonna (max 33) per la posizione PRINT.
X1	5C89		Il numero di linee (max 24) per la posizione PRINT.
X2	5C8A	SPOSNL	Come SPOSN, ma per la parte inferiore.
1	5C8C	SCR CT	Serve da contatore per lo "scroll"; è sempre

nº di byte	Indirizzo esadecimale	Nome	Funzione
			maggiore di uno del numero di "scroll" che verranno effettuati prima di fermarsi con il messaggio "scroll?". Se in questa variabile viene messo, con POKE come sempre, il valore 255, lo scroll continuerà senza interruzione.
1	5C8D	ATTR P	Gli attributi permanenti (fissati dalle istruzioni globali INK, PAPER etc.).
1	5C8E	MASK P	Usato per gli attributi trasparenti. Ogni bit pari a uno indica che l'attributo corrispondente non va preso da ATTR P, ma da ciò che già si trova sul video.
1	5C8F	ATTR T	Gli attributi temporanei correnti (fissati per esempio con le istruzioni PRINT, PLOT, DRAW etc.).
1	5C90	MASK T	Come MASK P, ma temporaneo.
1	5C91	P FLAG	Altri segnali.
30	5C92	MEMBOT	Queste variabili servono per permettere al calcolatore di memorizzare in "memorie" speciali sei diversi numeri a virgola mobile con cinque byte ciascuno.
2	5CB0	INTERR*	Il vecchio vettore di "interrupt", non utilizzato per via di una caratteristica dei programmi della ROM.
2	5CB2	RAMTOP	L'indirizzo dell'ultimo byte di area BASIC.
2	5CB4	PRAMT	L'indirizzo dell'ultimo byte di RAM effettiva.

APPENDICE D

LE MNEMONICHE DELLO Z80

In questa appendice viene fornita una breve descrizione per ciascuno dei comandi dello Z80: viene spiegata in dettaglio l'operazione svolta e come vengono modificati i segnali.
Non vengono riportati i codici esadecimali ma li si può trovare nell'appendice A, dove tutte le mnemoniche per lo Z80 sono elencate con i loro codici operativi ed i codici dell'ASCII Spectrum e dell'ASCII standard ad essi associati.

ALCUNE ABBREVIAZIONI:

r = registro ad un solo byte: A,B,C,D,E,H oppure L.

nn = un byte singolo di dati.

nnnn = due byte di dati.

d1 = un registro a due byte: BC, DE, HL oppure SP.

d2 = un registro a due byte: BC, DE, HL, IX, IY oppure SP.

dis = byte di "displacement" (spostamento), espresso secondo la convenzione del complemento a due.

x = bit numero 0,1,2,3,4,5,6 o 7.

res = un valore esadecimale ad un solo byte: 00, 08, 10, 18, 20, 28, 30 oppure 38.

ADC A,r : Aggiunge il registro r all'accumulatore, e vi somma anche il valore del segnale di riporto all'inizio dell'operazione. Con l'eccezione di ADC A,A, il contenuto del registro operando non viene modificato. Il segnale N viene posto a zero da ADC e gli altri segnali rispecchieranno lo stato finale dell'accumulatore.

ADC A,(HL); : Istruzione analoga alla ADC A,r, tranne per il fatto

ADC A,(IX + dis); ADC A,(IY + dis); ADC A,nn		che all'accumulatore viene sommato un byte puntato da (HL), (IX + dis), (IY + dis) oppure fornito direttamente.
ADC HL,d1	:	Compie l'addizione a due byte, sommando prima il byte di riporto al bit meno significativo del registro L, e poi sommando al registro HL il registro a due byte di (BC, DE, HL o SP). Il segnale N viene posto a zero e gli altri segnali rispecchieranno lo stato finale di HL.
ADD A,r	:	Compie una semplice addizione ad un solo byte, sommando il registro r all'accumulatore. ADD pone a zero il segnale N, mentre gli altri segnali rispecchieranno lo stato dell'accumulatore.

ADD A,(HL);
ADD A,(IX + dis);
ADD A,(IY + dis);
ADD A,nn
	:	Esattamente come ADD A,r, tranne che invece del registro r viene sommato il byte di dati puntato da (HL), (IX + dis) oppure fornito direttamente.

ADD HL,d1;
ADD IX,d1;
ADD IY,d1
	:	Viene eseguita un'addizione a due byte ai registri HL, IX e IY rispettivamente. Un registro a due byte d1 (ovvero HL, BC, DE o SP) viene sommato e rimane inalterato alla fine dell'operazione (tranne nel caso ADD HL,HL).

AND r	:	Esegue l'operazione logica AND sull'operatore. I bit del registro r vengono confrontati con quelli dell'accumulatore: per ogni bit pari a 1 sia nell'accumulatore che nel registro r, il bit corrispondente nell'accumulatore viene mantenuto ad 1, altrimenti viene posto a zero. Il registro operando r non viene alterato con questa istruzione. I segnali vengono opportunamente fissati.
AND (HL); AND(IX + dis); AND(IY + dis); AND nn	:	Viene eseguita l'operazione logica AND sull'accumulatore. Il dato su cui puntano (HL), (IX + dis), (IY + dis) oppure direttamente fornito, viene confrontato con AND con l'accumulatore. Il risultato viene lasciato nell'accumulatore, mentre l'operando (il byte (HL), (IX + dis), (IY + dis) oppure nn) non viene alterato. I segnali vengono fissati secondo lo stato dell'Accumulatore.

BIT x,r;
BIT x(HL);
BIT x,(IX + dis);
BIT x,(IY + dis)
: Questa istruzione serve per sottoporre a test il bit x (dove x è un numero fra 0 e 7) del registro r oppure della locazione (HL), (IX + dis) o (IY + dis). Se il bit è zero, il segnale di azzeramento viene posto a uno; se il bit è uno il segnale viene posto a zero. Il segnale C non viene modificato, il segnale H viene posto a uno e il segnale N viene azzerato. Lo stato dei segnali S e P/V non è definibile a priori.

CALL nnnn
: Provoca un salto alla subroutine posta alla locazione nnnn. L'indirizzo dell'istruzione successiva viene messo in uno stack e si rimane lì fino a quando non si incontra un'istruzione RET. I vari segnali non sono influenzati dall'istruzione CALL.

CALL C,nnnn;
CALL M,nnnn;
CALL NC,nnnn;
CALL NZ,nnnn;
CALL P,nnnn;
CALL PE,nnnn;
CALL PO,nnnn
CALL Z,nnnn
: Queste istruzioni operano esattamente nello stesso modo della CALL incondizionata, tranne che esse saranno ignorate a meno che la condizione non risulti soddisfatta. I segnali non sono influenzati.

CCF
: La condizione del segnale di riporto viene rovesciata rispetto al suo stato corrente. Il segnale N viene posto a zero e gli altri segnali non vengono modificati.

CP r
: Questa istruzione sottrae il registro r dall'accumulatore, senza però modificare effettivamente l'accumulatore o il registro r; i segnali invece vengono fissati in modo da riflettere il risultato dell'operazione.

CP (HL);
CP (IX + dis);
CP (IY + dis);
CP nn
: Queste istruzioni funzionano esattamente come CP r, eccetto che, invece che con il registro r, l'accumulatore viene confrontato con il dato contenuto in (HL), (IX + dis) o (IY + dis), oppure fornito direttamente. I segnali rispecchiano il risultato della sottrazione.

CPD
: Il contenuto della locazione di memoria su cui è puntato HL viene confrontato con il contenuto dell'accumulatore. Il segnale N viene posto a 1, gli altri segnali riflettono il risultato del confronto, tranne il segnale di riporto che resta inalterato. Il registro BC,

che funge da contatore di byte, viene decrementato di 1; anche la coppia di registri HL viene decrementata per puntare alla successiva locazione, più in basso. Se il registro BC diventa zero, il segnale P/V viene azzerato, altrimenti esso viene posto a 1.

CPDR : Questa funziona esattamente come CPD, tranne che dopo aver decrementato HL e BC, se BC non è nullo l'operazione di confronto viene ripetuta. Essa non viene ripetuta se BC è zero oppure se il valore dell'accumulatore è uguale a quello nella locazione di memoria su cui punta HL; in altre parole la si ripete fino a trovare un valore uguale.

CPI : Questa funziona esattamente come CPD, tranne per il fatto che il registro HL viene incrementato invece che decrementato.

CPL : Il contenuto dell'accumulatore viene complementato a due: i bit pari a uno vengono posti a zero e quelli pari a zero vengono posti a uno. Tutti i segnali della CPU non vengono modificati, tranne H e N che vengono posti a uno.

DAA : Questa istruzione modifica il valore dell'accumulatore dal suo valore binario a due cifre decimali con codifica binaria (BCD): i quattro bit più significativi rappresentano la cifra delle "decine" e i quattro bit meno significativi rappresentano le "unità".

DEC r : Questa istruzione riduce di uno il contenuto del registro r. Essa non modifica il segnale C, mentre il segnale N viene azzerato. Gli altri segnali riflettono il valore risultante nel registro r.

DEC (HL);
DEC (IX + dis);
DEC (IY + dis) : Questi comandi riducono di uno il contenuto della locazione su cui è puntato (HL), (IX + dis) o (IY + dis). I segnali vengono modificati nel modo descritto precedentemente per DEC r.

DEC d2 : Questa istruzione decrementa di uno il registro a due byte d2 (BC, DE, HL, SP, IX oppure IY). Essa non modifica i segnali.

DI : Questo comando dà ordine alla CPU di non accettare gli interrupt mascherabili.

DJNZ dis : Con questo comando il registro B viene decrementato di uno. Se B non risulta pari a zero, viene effettuato un salto relativo, usando dis e il complemento a due: dis viene sommato al "contatore di programma" (Program Counter).

E1 : Il comando di abilitazione degli interrupt, che da istruzione alla CPU di accettare gli interrupt mascherabili. Dopo che un interrupt è stato ricevuto, non ne verranno accettati altri fino a quando non si incontri il successivo comando E1.

EX AF,AF' : Questo comando da un solo byte fa sì che i valori dei registri AF e AF' si scambino di posto. I segnali vengono posti in modo da riflettere il contenuto dei segnali del nuvo gruppo dopo lo scambio.

EXX : I registri a due byte, BC, DE, e HL, vengono scambiati di posto con i loro corrispondenti nel gruppo uno.

EX DE,HL : Questa istruzione fa scambiare tra loro i contenuti dei registri a due byte DE e HL.

EX (SP),HL : Il valore in cima allo stack viene scambiato con HL da questa istruzione. SP e i segnali non ne risultano modificati.

EX (SP),IX;
EX (SP),IY : Analoghe a EX (SP),HL, queste due istruzioni scambiano con HL il contenuto dei registri IX e IY rispettivamente. I segnali non vengono modificati.

HALT : Questa istruzione provoca l'arresto di tutte le operazioni della CPU, e si rimane in attesa di un interrupt o di un segnale di reset. Il rinfresco della memoria di tipo dinamico (com'è quella dello Spectrum) è assicurato, poiché il processore esegue a questo scopo una serie di istruzioni NOP, che non alterano in alcun modo segnali o registri.

IMØ : Questa istruzione fissa a zero il modo di interrupt. Quando riceve un interrupt, la CPU permette ad un dispositivo interno appositamente progettato ed

attivato di inviarle un istruzione sul bus dei dati. La CPU poi eseguirà quell'istruzione. I segnali non vengono modificati da nessuna delle istruzioni IM.

IM1	: Questa istruzione pone a uno il modo di interrupt. Ricevuto un interrupt, la CPU esegue un RESTART alla locazione 38 Hex. I segnali della CPU non vengono modificati.
IM2	: Questa pone a due il modo di interrupt. Quando riceve un interrupt la CPU passa alla locazione il cui indirizzo è formato dal registro del vettore di interrupt (IV, o semplicemente I) per gli otto bit più significativi, e dall'informazione posta nel bus di dati per gli otto bit meno significativi.
IN r,(C)	: Questo comando fa sì che venga letto il valore in ingresso dal porto C, e che lo si metta nel registro r. I segnali della CPU non vengono alterati da questo comando.
IN A,nn	: Funziona esattamente nello stesso modo del precedente, tranne che viene usato direttamente un byte nn di dati ed il registro A invece dei registri C ed r rispettivamente.
INC r	: Questa istruzione incrementa di uno il valore del registro r. Il segnale di riporto non viene modificato, il segnale N viene azzerato, e gli altri segnali rispecchiano il valore risultante nel registro r.
INC (HL); INC (IX + dis); INC (IY + dis)	: Questo comando incrementa di uno il valore contenuto nella locazione su cui è puntato (HL), (IX + dis) o (IY + dis). I segnali vengono influenzati nel modo descritto per INC r.
INC d2	: Questa istruzione incrementa di uno il valore del registro a due byte d2 (cioè BC, DE, HL, SP, IX o IY). Essa non ha influenza sui vari segnali.
IND	: Questo comando fa sì che venga ricevuto un byte di dati dalla porta di input specificata dal registro C. Il dato viene trasferito nella locazione di memoria puntata da HL, poi HL viene decrementato di uno. Il registro B, che funge da contatore, viene ridotto

	anch'esso di uno e se così facendo B si azzera il segnale Z viene posto a uno. Lo stato dei segnali S,M e P/V non può essere predeterminato. Il segnale N viene sempre posto a uno da questo comando, mentre il segnale di riporto non viene toccato.
INDR	: Questa istruzione funziona esattamente come la IND descritta prima, però il registro B non è nullo alla fine di un'operazione, l'operazione stessa di lettura di un byte di dati viene ripetuta. In questo modo alla fine dell'operazione i segnali saranno come detto prima, tranne che per il registro Z che sarà posto a uno.
INI	: Questo comando opera esattamente come IND, però la coppia di registri HL viene incrementata di uno invece che decrementata.
INIR	: Questo funziona esattamente come INDR, solo che HL viene incrementato.
JP(HL)	: Questo comando fa eseguire un salto all'indirizzo specificato nel registro a due byte HL.
JP(IX); JP(IY)	: Questo comando provoca un salto all'indirizzo specificato dal registro indice IX o IY. I segnali non vengono modificati.
JP nnnn	: Questo provoca un salto direttamente all'indirizzo nnnn. I segnali non vengono modificati da questo comando.
JP C,nnnn; JP M,nnnn; JP NC,nnnn; JP NZ,nnnn; JP P,nnnn; JP PE,nnnn; JP PO,nnnn; JP Z,nnnn	: Questi comandi vengono eseguiti dalla CPU soltanto se la condizione risulta soddisfatta. In tal caso viene effettuato un salto diretto all'indirizzo nnnn. I segnali non vengono influenzati da questi comandi.
JR dis	: Questa istruzione fa eseguire un salto relativo. L'indirizzo di destinazione viene fornito usando il byte di spostamento dis e la convenzione del complemento a due. Lo spostamento viene calcolato dall'indirizzo

dell'istruzione successiva. JR lascia inalterati i segnali.

JR C,dis; JR NC,dis; JR NZ,dis; JR Z,dis	:	Questi comandi operano esattamente come JR dis, eccetto che essi vengono eseguiti dalla CPU soltanto se la condizione è soddisfatta.

LD (nnnn),A
LD (nnnn),d2
LD (BC),A
LD (DE),A
LD (HL),r
LD (HL),nn
LD (IX + dis),r
LD (IX + dis),nn
LD (IY + dis),r
LD (IY + dis),nn
LD A,(nnnn)
LD A,(BC)
LD A,(DE)
LD r,(HL)
LD r,(IX + dis)
LD r,(IY + dis)
LD r,(r)
LD r,nn
LD d2,(nnnn)
LD d2,nnnn
LD A,I
LD I,A
LD A,R
LD R,A
LD SP,HL
LD SP,IX
LD SP,IY
	:	Il comando ha un incredibile numero di combinazioni: qui vengono elencate tutte le possibili forme sintattiche. Esso serve semplicemente a copiare il valore del termine di destra, che può essere il contenuto di una locazione di memoria, un registro singolo o a due byte oppure direttamente un numero, nel termine di sinistra, che a sua volta può essere un registro singolo o a due byte, oppure una locazione di memoria. Non tutte le combinazioni sono ammissibili, perciò quando scrivete un programma controllate che la forma sintattica che state usando compaia nella lista dell'Appendice A, per essere certi di non scrivere un programma impossibile...

LDD	:	Il contenuto della locazione di memoria su cui è puntato il registro HL viene trasferito nella locazione su cui è puntato DE. Poi DE e HL vengono ridotti di uno. Anche BC viene ridotto di uno, e se in questo modo BC diventa zero il segnale P/V viene posto a zero, altrimenti viene posto a uno. I segnali H e N vengono azzerati, mentre gli altri non vengono toccati.

LDDR : Questo comando funziona esattamente come LDD, tranne che se BC non è zero alla fine dell'operazione, lo stesso comando viene ripetuto. Ciò significa che quando l'esecuzione è terminata il segnale P/V sarà zero e gli altri segnali come descritto sopra.

LDI : Questo funziona proprio come LDD, tranne che sia HL che DE vengono incrementati invece che ridotti di uno. I segnali vengono influenzati in modo esattamente uguale.

LDIR : Questo comando funziona come LDDR tranne che anche qui sia HL che DE vengono incrementati invece che ridotti di uno.

NEG : Questa istruzione nega il contenuto dell'accumulatore secondo la convenzione del complemento a due. In pratica succede che prima tutti i bit vengono invertiti, cioè quelli che sono zero vengono posti a uno e viceversa: poi al risultato viene sommato uno. I segnali S e Z rispecchiano il risultato dell'operazione, mentre il segnale P/V viene posto a uno se il valore dell'accumulatore prima, e quindi anche dopo l'operazione, era 8Ø Hex; altrimenti esso viene posto a zero. Il segnale C viene azzerato se l'accumulatore conteneva, sia prima che dopo l'operazione, il valore ØØ Hex; altrimenti viene posto a uno. Il segnale N viene sempre posto a uno.

NOP : Questo comando fa semplicemente perdere del tempo alla CPU, senza farle fare altro che spostare in avanti il Program Counter alla locazione successiva! Nessun segnale viene modificato.

OR r : Il registro ad un solo byte r viene confrontato con l'operazione logica OR con il contenuto dell'accumulatore. In pratica si ha che i bit dell'accumulatore vengano confrontati uno alla volta con quelli corrispondenti del registro r. Se uno o entrambi i due bit messi a confronto corrispondono a uno, allora il bit dell'accumulatore viene posto a uno; altrimenti viene posto a zero, ed anche il segnale N. Il segnale H viene sempre posto a uno e gli altri segnali riflettono il contenuto dell'operazione.

OR (HL); OR (IX + dis); OR (IY + dis); OR nn	:	Allo stesso modo descritto per OR r, il contenuto della locazione di memoria su cui è puntato (HL), (IX + dis) oppure (IY + dis), o un dato direttamente fornito, viene confrontato con l'operazione logica OR con l'accumulatore.
OTDR	:	Analogamente a quanto descritto per INDR, i dati vengono trasferiti dalla locazione di memoria puntata da HL alla porta specificata nel registro C. I vari segnali non vengono modificati.
OTIR	:	Questo comando opera esattamente come OTDR, solo che HL viene incrementato.
OUT (C),r	:	Con questo comando il dato contenuto nel registro r viene inviato in uscita alla porta I/O indicata dal registro C. I vari segnali non vengono modificati.
OUT nn,A	:	Questo comando è analogo ad OUT C,r, eccetto che il numero nn specifica la porta da usare ed il dato viene preso dal registro A.
OUTD	:	Questo è molto simile al comando IND precedentemente descritto, tranne che con OUTD il dato viene inviato in uscita alla porta della locazione indicata da HL.
OUTI	:	Questo comando funziona come OUTD, però dopo ogni operazione il registro HL viene incrementato invece che ridotto.
POP AF	:	Il numero a due byte in cima allo stack viene rimosso e posto nella coppia di registri AF. Il puntatore dello stack, SP, viene poi aumentato di due in modo da puntare al numero teoricamente sottostante. I segnali non vengono alterati.
POP BC; POP DE; POP HL; POP IX; POP IY	:	Questi comandi funzionano esattamente come POP AF, tranne per il fatto che le coppie di registri che ricevono il numero rimosso dallo stack sono rispettivamente BC, DE, HL, IX e IY.
PUSH AF	:	Questo comando ha un effetto opposto a POP AF.

Il valore contenuto nella coppia AF viene messo sullo stack e il puntatore dello stack, SP, viene ridotto di due. I segnali non vengono modificati.

PUSH BC;
PUSH DE;
PUSH HL;
PUSH IX;
PUSH IY
: Queste istruzioni funzionano come PUSH AF, però i valori che vengono messi sullo stack sono quelli dei registri BC, DE, HL, IX e IY rispettivamente.

RES x,r
: Questo comando pone a zero il bit x del registro ad un solo byte r. I segnali della CPU non risultano influenzati.

RES x,(HL);
RES x,(IX + dis);
RES x,(IY + dis)
: Queste istruzioni pongono a zero il bit x della locazione di memoria puntata da (HL), (IX + dis) o (IY + dis) rispettivamente.

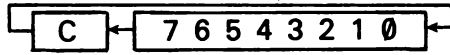
RET
: Il valore in cima allo stack viene rimosso e trasferito nel contatore di programma PC. Il puntatore dello stack viene aumentato di due, cosicché punta al dato teoricamente sottostante. L'esecuzione del programma continua usando il nuovo valore contenuto in PC. I segnali della CPU non vengono toccati.

RET C;
RET M;
RET NC;
RET NZ;
RET P;
RET PE;
RET PO;
RET Z
: Questi comandi operano esattamente come RET, ma vengono eseguiti solo se la condizione risulta soddisfatta. I segnali non vengono modificati.

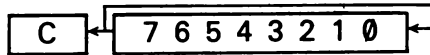
RETI;
RETN
: Non avrete direttamente bisogno di queste due istruzioni: esse si riferiscono al sistema di interrupt dello Z8Ø. Nello Spectrum gli interrupt vengono usati per la scansione della tastiera. In alcune circostanze questa scansione viene esclusa, per esempio quando è in funzione la stampante o quando si sta usando l'interfaccia per il nastro. Si veda il capitolo otto.

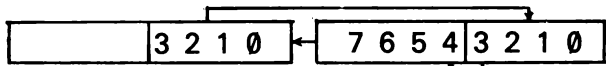
RL r;
RL (HL);
: I bit del registro r o della locazione di memoria su cui puntano (HL), (IX + dis) o (IY + dis) vengono

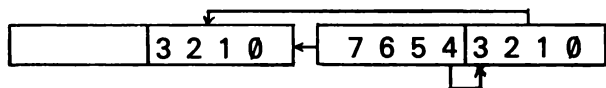
RL (IX + dis);
RL (IY + dis)

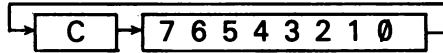
fatti ruotare insieme al bit di riporto, come indicato nel disegno che segue:

I segnali H ed N vengono azzerati, mentre i segnali S, Z e P/V riflettono il risultato dell'operazione. Il segnale P/V riflette lo stato di parità del registro o della locazione di memoria dopo la rotazione.

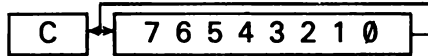
RLC r;
RLC (HL);
RLC (IX + dis);
RLC (IY + dis)

: Il registro r, o la locazione di memoria puntata da (HL), (IX + dis) o (IY + dis) rispettivamente, viene ruotato nel modo indicato dal disegno sotto. I segnali si comportano come per l'istruzione RL r.

RLD

: Questa istruzione consente di ruotare verso sinistra un numero decimale a codifica binaria (BCD) a quattro bit posto nella metà meno significativa dell'accumulatore con due cifre BCD nella locazione di memoria indicata da HL. Ciò viene chiarito dal disegno seguente:

RRD

: Questo comando è molto simile al precedente, perché fa anch'esso ruotare alcune cifre BCD. In questo caso però le cifre vengono ruotate verso destra, come nel seguente disegno:

RR r;
RR (HL)

: Questi comandi funzionano in modo analogo a RL r, tranne per il fatto che la rotazione avviene in direzione contraria, come indicato sotto:

RRC r;
RRC (HL);
RRC (IX + dis);
RRC (IY + dis)

: Questi comandi funzionano in modo analogo a RLC r, tranne per il fatto che la rotazione avviene in direzione contraria, come indicato sotto:

RST res

: Questo comando è in pratica una versione abbreviata dell'istruzione CALL, poiché fa eseguire la routine posta all'indirizzo specificato con res; vi è però una restrizione sui possibili valori di res, che può essere soltanto 00, 08, 10, 18, 20, 28, 30 o 38. RST è l'abbreviazione di RESTART. Questo comando non ha influenza sui registri.

SBC A,r

: Con questo comando il registro r viene sottratto dall'accumulatore. Il contenuto del segnale di riporto viene sottratto dal bit meno significativo dell'accumulatore. Il segnale N viene posto a uno, mentre gli altri segnali non vengono toccati.

SBC A,nn;
SBC A,(HL);
SBC A,(IX + dis);
SBC A,(IY + dis)

: Questa istruzione funziona esattamente come SBC A,r, però dall'accumulatore viene sottratto il dato nn fornito direttamente, oppure il contenuto delle locazioni di memoria (HL), (IX + dis), (IY + dis).

SBC HL,d2

: Questa istruzione compie una sottrazione a due byte con il segnale di riporto. Il registro a due byte d2 viene sottratto da HL, e poi anche il segnale di riporto viene sottratto dal bit meno significativo di HL. Il segnale N vien posto a uno, mentre gli altri segnali rispecchiano il risultato in HL.

SCF

: Con questo comando viene semplicemente posto a uno il segnale di riporto. I segnali H ed N vengono azzerati, tutti gli altri non vengono modificati.

SET x,r

: Questo comando pone a uno il bit x del registro r. Nessun segnale della CPU viene modificato.

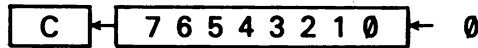
SET x,(HL);

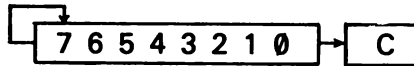
: Queste istruzioni pongono il bit x a uno nella locazio-

SET x,(IX + dis);
SET x,(IY + dis)

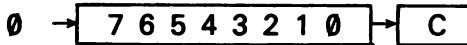
ne di memoria indicata da (HL), (IX + dis), (IY + dis) rispettivamente. Nessuno dei segnali viene modificato.

SLA r;
SLA (HL);
SLA (IX + dis);
SLA (IY + dis)

: Questi comandi fanno spostare verso sinistra il registro r oppure il contenuto della locazione indicata da (HL), (IX + dis) o (IY + dis), come indicato nel disegno che segue. I segnali si comportano come descritto per RL r.

SRA r;
SRA (HL);
SRA (IX + dis);
SRA (IY + dis)

: Questi comandi sono simili a quelli descritti per SLA, però lo spostamento viene fatto nella direzione opposta, come indicato nel disegno seguente.

SRL r;
SRL (HL);
SRL (IX + dis);
SRL (IY + dis)

: Queste istruzioni fanno spostare verso destra il registro r, oppure il contenuto della locazione indicata da (HL), (IX + dis) o (IY + dis), rispettivamente, come nel disegno che segue. I segnali vengono modificati come descritto per RL.

0 → 7 6 5 4 3 2 1 0 → C

SUB r

: Questo comando esegue una semplice sottrazione ad un solo byte: dall'accumulatore viene sottratto il registro r, che però non viene modificato. Il segnale N viene posto a uno, mentre gli altri segnali rispecchiano il valore finale nell'accumulatore.

SUB (HL);
SUB (IX + dis);
SUB (IY + dis)
SUB nn

: Queste istruzioni funzionano esattamente come SUB r, tranne che, invece di sottrarre il registro r, dall'accumulatore viene sottratto il contenuto della locazione su cui è puntato (HL), (IX + dis) o (IY + dis), oppure direttamente il numero nn.

XOR r

: Il registro r viene confrontato con l'accumulatore mediante l'operazione logica OR esclusivo. Considerando un bit alla volta, se il bit nell'accumulatore e il corrispondente bit sono entrambi zero e entrambi uno, il bit nell'accumulatore viene posto a zero; altri-

menti esso viene posto a uno. Il segnale di riporto viene posto a zero come pure il segnale N, il segnale H viene posto a uno e gli altri segnali riflettono il valore risultante nell'accumulatore.

XOR nn;
XOR (HL);
XOR (IX + dis);
XOR (IY + dis)
: Questi comandi operano esattamente come il precedente, però il confronto di OR esclusivo con l'accumulatore viene fatto usando direttamente il numero nn, oppure usando il contenuto della locazione specificata da (HL), (IX + dis) o (IY + dis) rispettivamente.

Seconda tappa nel mondo del linguaggio macchina dello Spectrum, il libro riprende la chiarezza di linguaggio e il metodo del primo volume pur trattandosi, in realtà, di un testo slegato dal precedente. Rivolto a programmatori avanzati, vengono trattati in modo più specifico gli argomenti quali i salti, salti relativi, flag, AND, OR e XOR, cicli, registri, ..., il tutto partendo dai fondamenti per arrivare ad una buona conoscenza del codice macchina.

L. 13.000

COD CC221 ISBN 88-7056-287-5

221
IL LINGUAGGIO MACCHINA DELLO SPECTRUM VOL. 2
Paul Holmes
GRUPPO
EDITORIALE
JACKSON